

AFRL-IF-RS-TR-2002-299
Final Technical Report
November 2002



COMPUTE-INTENSIVE METHODS AND HYBRID APPROACHES FOR COMBINATORIAL PROBLEMS

Cornell University

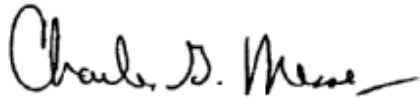
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK**

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2002-299 has been reviewed and is approved for publication.

APPROVED:



CHARLES G. MESSENGER
Project Engineer



FOR THE DIRECTOR:

MICHAEL L. TALBERT, Maj., USAF
Technical Advisor, Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE			<i>Form Approved</i> <i>OMB No. 074-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE NOVEMBER 2002	3. REPORT TYPE AND DATES COVERED Final Mar 99 – Jul 01	
4. TITLE AND SUBTITLE COMPUTE-INTENSIVE METHODS AND HYBRID APPROACHES FOR COMBINATORIAL PROBLEMS			5. FUNDING NUMBERS C - F30602-99-1-0005 & F30602-99-1-0006 PE - 62702F, 61102F PR - 2304 TA - GC WU - P1, P2	
6. AUTHOR(S) Carla Gomes				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Cornell University 120 Day Hall Ithaca New York 14853			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFTB 525 Brooks Road Rome New York 13441-4505			10. SPONSORING / MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2002-299	
11. SUPPLEMENTARY NOTES AFRL Project Engineer: Charles G. Messenger/IFTB/(315) 330-3528/ Charles.Messenger@rl.af.mil				
12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.				12b. DISTRIBUTION CODE
13. ABSTRACT (Maximum 200 Words) Our research program focuses on techniques that lie at the intersection of Artificial Intelligence and Operations Research. In particular, we study computational methods for large-scale combinatorial optimization. Our research combines formal analysis and design of optimization techniques with the study of applications such as planning and scheduling, autonomous distributed agents and combinatorial auctions. Central themes of our work are (1) the integration of concepts from mathematical programming with constraint programming, (2) the study of the impact of structure on problem hardness, and (3) the use of randomization techniques to improve the performance of exact (complete) search methods. This report highlights some of our research projects and accomplishments.				
14. SUBJECT TERMS Combinatorial Optimization, Search, Integration of AI/OR Techniques, Randomized Methods, Approximations, CSP, Distributed CSP			15. NUMBER OF PAGES 56	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

Table of Contents

1	Introduction	1
2	Integration of Methods from Artificial Intelligence and Operations Research	1
3	Randomized Complete Search Techniques and Heavy-Tailed Distributions	2
4	Synopsis of Papers in the Appendix	4
	References	5
Appendix 1	The Promise of LP to Boost CSP Techniques for Combinatorial Problems	7
Appendix 2	Formal Models of Heavy-Tailed Behavior in Combinatorial Search	23
Appendix 3	Communication and Computation in DisCSP Algorithms	38

Hybrid Compute-Intensive Approaches for Combinatorial Problems

Carla P. Gomes
Computer Science Department
Cornell University
Ithaca, NY 14853
gomes@cs.cornell.edu

1 Introduction

This is the final report for contract F30602-99-1-0005 (Compute-Intensive Methods for Combinatorial Problems) and F30602-99-1-0006 (Hybrid Approaches for Combinatorial Problems). Our research under these contracts focused on techniques that lie at the intersection of Artificial Intelligence and Operations Research. In particular, we considered computational methods for large-scale combinatorial optimization. Our research combines formal analysis and design of optimization techniques with the study of applications such as planning and scheduling, autonomous distributed agents, and combinatorial auctions. Central themes of our work are (1) the integration of concepts from mathematical programming with constraint programming, (2) the study of the impact of structure on problem hardness, and (3) the use of randomization techniques to improve the performance of exact (complete) search methods.

We now highlight some of our research projects and research accomplishments. As an appendix to this report we include three representative papers that elaborate on the topics.

2 Integration of Methods from Artificial Intelligence and Operations Research: Bridging Constraint Programming and Mathematical Programming

A key focus of our research program is on combining constraint programming techniques with concepts from mathematical programming, drawing on the individual strengths of these paradigms. For example, in recent work, we have

developed a hybrid approach for combinatorial optimization, consisting of an exact randomized complete search method that tightly couples constraint propagation techniques with information obtained from randomized rounding of linear programming (LP) relaxations [7]. This hybrid strategy outperforms pure constraint programming and LP strategies and other approaches. The hybrid approach relies on the LP relaxation to provide good approximate solutions. For the quasigroup completion problem, a combinatorial problem with structural properties similar to those of a variety real-world optimization problems, we developed a new approximation that is within a factor of $1 - 1/e \approx 0.63$ from optimal [6]. The best earlier approximations gave a factor of 0.5. Our approximation uses randomized LP rounding, based on an LP relaxation of a packing formulation of the problem.

More generally, we have shown tradeoffs between problem representations based on constraint programming, logical representations (Boolean satisfiability encodings), and mathematical programming formulations. Problem representation is a key factor affecting the efficiency of combinatorial search. For example, we have shown how one can substantially improve the performance of search methods by using primal-dual encodings, by adding inferred (redundant) constraints, as well as by exploiting the well-structured subcomponents of problems with efficient propagation algorithms [3, 4, 5].

For a presentation covering recent results:

www.cs.cornell.edu/gomes/cpaior02.ppt

See also the paper in Appendix 1 for a description of the hybrid approach for combinatorial optimization mentioned above, combining an exact randomized complete search method with constraint propagation techniques using the information obtained from randomized rounding of linear programming (LP) relaxations [7].

3 Randomized complete search techniques and heavy-tailed distributions

Randomized search strategies have been highly successful in local search (e.g., simulated annealing [14], tabu search (Glover 1989), and genetic algorithms (Holland 1975)). However, such methods are inherently incomplete, in that they do not guarantee optimality of the solution. Optimality can be guaranteed using backtrack style methods such as branch-and-bound. These methods can explore the full combinatorial space. Backtrack style search can be randomized by introducing a random element in the variable choice and / or value selection heuristics. With some minimal additional book keeping, one can still maintain completeness of the search strategy. Researchers have observed that the performance of backtrack search methods can vary considerably from instance to

instance. There have been some theoretical results showing that randomization can improve the performance of complete search methods, *e.g.*, [18, 15], but randomization was not believed to provide significant practical benefits in a complete search setting. For example, up to about five years ago state-of-the-art Davis-Putnam style Boolean satisfiability solvers did not include randomization. In our work on the study of the run time distribution of complete search methods, we have demonstrated that one can in fact obtain exponential speedups by randomizing a complete search method [10]. Such speedups can be obtained by taking advantage of the high variance in run time of randomized complete methods. In particular, we have shown that the extreme variance or *unpredictability* in the run time of complete search procedures on combinatorial problems can often be explained by the phenomenon of heavy-tailed distributions. Heavy-tailed distributions are highly non-standard probability distributions, capturing phenomena with infinite moments, for example infinite variance or infinite mean. Previously such distributions have been used to model erratic behavior in, for example, weather patterns, stock market behavior, and time delays on the World Wide Web. In more recent work, we have developed formal models of backtrack search that provably exhibits heavy-tailed behavior [1]. The understanding of the extreme variance that characterizes complete search algorithms on combinatorial problems has led to the introduction of novel strategies for the design of algorithms based on “rapid restarts” and “portfolio” strategies [12, 10, 2]. In a restart strategy, one repeatedly restarts the search procedure with a new random seed after a certain predefined number of backtracks; in an algorithm portfolio, many copies of a randomized search procedure (each started with a different random seed) or a mix of different search procedures are executed in parallel or interleaved. Restarts and portfolio can significantly reduce the variance in run time and the probability of failure of the search procedures, resulting in more robust and more efficient overall search methods. We have shown that restarts provably eliminate heavy-tailed behavior. The results of this research have changed the general view of randomization of complete search methods in, for example, the Satisfiability and Constraint satisfaction community. Randomization and restart strategies have now been incorporated into state-of-the-art solvers for the Boolean satisfiability problem (SAT solvers). In fact, the rapid restarts technique is an integral component of the current world’s fastest SAT solver, called Chaff [17]. (In Chaff, restarting is combined with clause learning, another technique central to Chaff’s overall effectiveness. In clause learning, certain derived clauses are carried over between restarts.) Chaff can handle problem instances with over one million variables and four million constraints. The solver has been used to verify correctness properties of the latest Alpha chip design, which has a complexity comparable to that of the Pentium IV. We have also demonstrated the effectiveness of randomization and restarts for branch-and-bound search methods, distributed constraint satisfaction, and planning and scheduling problems. Randomized restarts have been demonstrated to be effective for reducing total execution time on a wide variety of problems in

scheduling, theorem proving, circuit synthesis, planning, and hardware verification [12, 10, 9, 2, 16, 17]. In our current research we are investigating optimal policies for restart strategies [8, 11, 13].

4 Synopsis of the papers in the Appendix

In the Appendix we include three papers that elaborate on several issues discussed above. In this section we give a short synopsis of each paper.

The Promise of LP to Boost CSP techniques for Combinatorial Problems In this paper we propose a complete randomized backtrack search method for combinatorial problems that tightly couples CSP propagation techniques with randomized LP rounding. The approach draws on recent results on approximation algorithms with theoretical guarantees, based on LP relaxations and randomized rounding techniques, as well on results that provide evidence that the run time distributions of combinatorial search methods are often heavy-tailed. We present experimental results that show that our hybrid CSP/LP backtrack search method outperforms the pure CSP and pure LP strategies on instances of a hard combinatorial problem.

Formal Models of Heavy-Tailed Behavior in Combinatorial Search (with Hubie Chen and Bart Selman) In this paper we discuss formal models of heavy-tailed behavior in combinatorial search.

Communication and Computation in DisCSP Algorithms In this paper we introduce a distributed benchmark based on a real-world application that arises in the context of networked distributed systems. In order to study the performance of Distributed CSP algorithms (DisCSP) in a truly distributed setting, we use a discrete-event network simulator, which allows us to model the impact of different network traffic conditions on the performance of the algorithms. We consider two complete DCSP algorithms: asynchronous backtracking (ABT) and asynchronous weak commitment search (AWC). In our study of different network traffic distributions, we found that, random delays, in some cases combined with a dynamic decentralized restart strategy, can improve the performance of DCSP algorithms. More interestingly, we also found that the *active introduction of message delays by agents can improve performance and robustness, while reducing the overall network load*. Finally, our work confirms that AWC performs better than ABT on satisfiable instances. However, on unsatisfiable instances, the performance of AWC is considerably worse than ABT.

References

- [1] H. Chen, C. Gomes, and B. Selman. Formal models of heavy-tailed behavior in combinatorial search. In *Proceedings of 7th Intl. Conference on the Principles and Practice of Constraint Programming (CP-2001), Lecture Notes in Computer Science, Vol. 2239, Springer-Verlag*, pages 408–422, 2001.
- [2] C. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126:43–62, 2001.
- [3] Carla Gomes. editor, *The Knowledge Engineering Review. Special Issue on the Integration of Artificial Intelligence and Operations Research Techniques, Vol. 15 (1) and Vol. 16 (1)*. Cambridge Press, 2000-2001.
- [4] Carla Gomes. editor, *The Knowledge Engineering Review. Special Issue on the Integration of Artificial Intelligence and Operations Research II, Vol. 16 (1)*. Cambridge Press, 2001.
- [5] Carla P. Gomes. Structure and Randomization: Common Themes in AI and OR. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-00)*, New Providence, RI, 2000. AAAI Press.
- [6] Carla P. Gomes, Rommel G. Regis, and David B. Shmoys. An Improved Approximation Algorithm for the Partial Latin Square Extension Problem. In *To appear in Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-03)*, 2003.
- [7] Carla P. Gomes and David Schmoys. The promise of LP to boost CSP techniques for combinatorial problems. In Narendra Jussien and François Laburthe, editors, *Proceedings of the Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'02)*, pages 291–305, Le Croisic, France, March, 25–27 2002.
- [8] Carla P. Gomes and Bart Selman. Hybrid Search Strategies for Heterogeneous Search Spaces. *International Journal on Artificial Intelligence Tools*, 2000.
- [9] Carla P. Gomes, Bart Selman, Nuno Crato, and Henry Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. of Automated Reasoning*, 24(1-2):67–100, 2000.
- [10] Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting Combinatorial Search Through Randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, New Providence, RI, 1998. AAAI Press.

- [11] E. Horvitz, Y. Ruan, C. Gomes, H. Kautz and B. Selman, and M. Chickering. A Bayesian Approach to Tackling Hard Computational Problems. In *Proceedings of the Seventeenth Conference On Uncertainty in Artificial Intelligence (UAI-01)*, 2001.
- [12] B. Huberman, R. Lukose, and T. Hogg. An economics approach to hard computational problems. *Science*, (265):51–54, 1993.
- [13] H. Kautz, E. Horvitz, Y. Ruan, C. Gomes, and B. Selman. Dynamic Restart Policies. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, Edmonton, Canada, 2002. AAAI Press.
- [14] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, (220):671–680, 1983.
- [15] M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of las vegas algorithms. *Information Process. Lett.*, pages 173–180, 1993.
- [16] Andreas Meier, C. Gomes, and E. Mellis. An Application of Randomization and Restarts to Proof Planning. In *Proceedings of the Sixth European Conference On Planning (ECP-01)*, 2001.
- [17] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conference*, pages 530–535, 2001.
- [18] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge University Press, 1995.

Appendix 1

The Promise of LP to Boost CSP Techniques for Combinatorial Problems*

The Promise of LP to Boost CSP Techniques for Combinatorial Problems*

Carla P. Gomes

Dept. of Comp. Science

Cornell University

Ithaca, NY 14853, USA

email: gomes@cs.cornell.edu

David B. Shmoys

Dept. of Comp. Science

School of Operations Research and Industrial Engineering

Cornell University

Ithaca, NY 14853, USA

email: shmoys@cs.cornell.edu

Abstract

In recent years we have seen an increasing interest in combining Constraint Satisfaction Problem (CSP) methods and Linear Programming (LP) techniques for solving hard computational problems. While considerable progress has been made in the integration of these techniques for solving problems that exhibit a mixture of linear and combinatorial constraints, it has been surprisingly difficult to successfully integrate LP-based and CSP-based methods in a purely combinatorial setting.

We propose a complete randomized backtrack search method for combinatorial problems that tightly couples CSP propagation techniques with randomized LP rounding. Our approach draws on recent results on approximation algorithms with theoretical guarantees, based on LP relaxations and randomized rounding techniques, as well on results that provide evidence that the run time distributions of combinatorial search methods are often heavy-tailed. We present experimental results that show that our hybrid CSP/LP backtrack search method outperforms the pure CSP and pure LP strategies on instances of a hard combinatorial problem.

*This research was partially funded by AFRL, grants F30602-99-1-0005 and F30602-99-1-0006, AFOSR, grant F49620-01-1-0076 (Intelligent Information Systems Institute) and F49620-01-1-0361 (MURI grant on Cooperative Control of Distributed Autonomous Vehicles in Adversarial Environments) and DARPA, F30602-00-2-0530 (Controlling Computational Cost: Structure, Phase Transitions and Randomization) and F30602-00-2-0558 (Configuring Wireless Transmission and Decentralized Data Processing for Generic Sensor Networks). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government.

1 Introduction

In recent years we have seen the development of successful methods for solving optimization problems by integrating techniques from Constraint Programming (CP) and Operations Research (OR) (see e.g., [8]). Such hybrid approaches draw on the individual strengths of these different paradigms: OR heavily relies on mathematical programming formulations such as integer and linear programming, while CP uses constrained-based search and inference methods. This is particularly true in domains where we have a combination of linear constraints, well-suited for linear programming (LP) formulations, and discrete constraints, suited for constraint satisfaction problem (CSP) formulations. Nevertheless, in a *purely combinatorial* setting, so far it has been surprisingly difficult to integrate LP-based and CSP-based techniques. For example, despite a significant amount of work on using LP relaxations to solve Boolean satisfiability (SAT) problems (see e.g., [11, 12, 16, 23]), practical state-of-the-art solvers do not incorporate LP relaxation techniques. From a practical point of view, the challenge is how to integrate such techniques into practical solvers. The basic idea is to use the information from LP relaxations to guide the combinatorial search process. A key issue is whether the LP relaxation provides sufficient useful additional information — in particular, information that is not easily uncovered by constraint propagation and inference techniques. Of course, also the cost of solving the LP relaxation should not outweigh the benefits in the reduction of search cost.

We propose a *complete* randomized backtrack search method that tightly couples CSP propagation techniques with randomized LP rounding. Our approach draws on recent results on some of the best approximation algorithms with theoretical guarantees based on LP relaxations and randomized rounding techniques (see e.g., [4, 19]), as well on results that uncovered the extreme variance or “unpredictability” in the run time of complete search procedures, often explained by the phenomenon of heavy-tailed cost distributions [10].

We use as a benchmark domain the quasigroup (or Latin square) completion problem (QCP). Each instance consists of an n by n matrix with n^2 cells. A complete quasigroup consists of a coloring of each cell with one of n colors in such a way that there is no repeated color in any row or column. Given a partial coloring of the n by n cells, determining whether there is a valid completion into a full quasigroup is an NP-complete problem [6]. The underlying structure of this benchmark is similar to that found in a series of real-world applications, such as timetabling, experimental design, and fiber optics routing problems [18, 17].

We present our preliminary experimental findings for our randomized hybrid CSP/LP backtrack search method on hard combinatorial instances of the QCP domain. We compare our results with a pure CSP strategy and with a pure LP strategy. Our results show that a hybrid approach does improve over the pure strategies. In our hybrid approach, the LP relaxation with rounding strategy provides global information about the values to assign to the CSP variables. In effect, the randomized LP rounding provides powerful heuristic guidance to the CSP search, at least at the top of the backtrack search tree. With our hybrid CSP/LP strategy we were able to considerably improve the time performance of the pure CSP strategy. Furthermore, the hybrid CSP/LP strategy could solve several instances of QCP that could not be solved by the pure CSP strategy. Interestingly, and contrarily to the experience in other domains that combine linear constraints

with a combinatorial component, we conjecture that the role of the LP relaxation in detecting infeasibility for pure combinatorial problems is not as important as its role as search heuristic. In particular, deeper down in the search tree, the information obtainable via LP relaxations can be computed much faster via CSP techniques. This means that during that part of the search process, the hybrid strategy should be avoided. A key issue in making the hybrid strategy effective is to find the right balance between the amount of work spent in solving the LP relaxations and the time spent on the CSP search. A detailed empirical and theoretical evaluation is currently under way. Our approach also uses restart strategies in order to combat the heavy-tailed nature of combinatorial search. By using restart strategies we take advantage of any significant probability mass early on in the distribution, reducing the variance in run time and the probability of failure of the search procedure, resulting in a more robust overall search method.

The structure of the paper is as follows. In the next section, we describe the Quasigroup Completion Problem (QCP). In section 3, we provide different formulations for the problem and, in section 4, we discuss approximations for QCP based on LP randomized rounding. In section 5, we present our hybrid CSP/LP randomized rounding backtrack search procedure and, in section 6, we provide empirical results.

2 The Quasigroup Completion Problem

A quasigroup is an ordered pair (Q, \cdot) , where Q is a set of n symbols and (\cdot) is a binary operation on Q such that the equations $a \cdot x = b$ and $y \cdot a = b$ are uniquely solvable for every pair of elements a, b in Q . The *order* n of the quasigroup is the cardinality of the set Q .

The best way to understand the structure of a quasigroup is to consider its n by n multiplication table, as defined by its binary operation: The constraints of a quasigroup are such that its multiplication table defines a *Latin Square*. A Latin Square is an $n \times n$ matrix on n symbols, such that each row/column is a permutation of its n symbols [18]. A *partial latin square PLS* is a partially filled n by n matrix such that no symbol occurs twice in a row or a column. $PLS_{i,j} = k$ denotes that entry i, j of PLS has symbol k . We refer to the empty cells of the partial latin square as *holes* and to the non-empty cells as *pre-assigned* cells. The number of holes in a *PLS* is denoted by h . The Quasigroup Completion Problem (QCP) (or Latin Square Completion Problem)¹ is the problem of determining whether the h holes of the corresponding partial latin square can be filled in such a way that we obtain a complete latin square (*i.e.*, a full multiplication table of a quasigroup) (see Figure 1):

	1	2	3
2		4	1
1	4		2
3		1	

4	1	2	3
2	3	4	1
1	4	3	2
3	2	1	4

Figure 1: Quasigroup Completion Problem of order 4, with 5 holes.

¹For simplicity, in the remaining of the paper, we will use quasigroup and latin square and partial quasigroup completion problem and partial latin square completion problem interchangeably.

QCP is an NP-complete problem [6]. We have identified a phase transition phenomenon for the completion problem [9]. At the phase transition, problem instances switch from being almost all solvable (“under-constrained”) to being almost all unsolvable (“over-constrained”). The computationally hardest instances lie at the phase transition boundary. Figure 2 shows the median computational cost and phase transition. Along the horizontal axis we vary the ratio of pre-assigned cells.² We note that even though all the instances are from an NP-complete problem, we clearly distinguish various regions of problem difficulty. In particular, both at low ratios and high ratios of preassigned colors the median solution cost is relatively small. However, in between these two regimes, the complexity peaks and, in fact, exhibits strong exponential growth.³

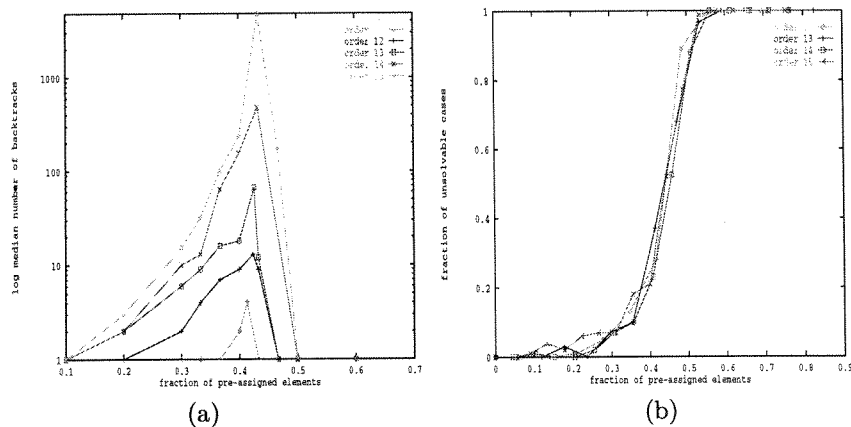


Figure 2: (a) Cost profile, and (b) phase transition for the quasigroup completion problem (up to order 15).

We identified the erratic behavior of the mean and the variance of different randomized backtrack style procedures running on instances of QCP [10]. Figure 3 (a) depicts this phenomenon, displaying the mean cost of a randomized backtrack style search procedure calculated over an increasing number of runs, on the *same* QCP instance, an instance of order 11 with 64% of holes. Despite the fact that this instance is easy, the median number of backtracks for solution is 1, some runs take more the 10^6 backtracks. Figure 3 (b) plots the log-log plot of the tail (*i.e.*, $(1-F(x))$) of the runtime distributions of a randomized backtrack search method on three QCP instances: one instance in the under-constrained area (the same instance of order 11 with 64% of holes), one in the critically constrained area, and one in the medium constrained area. The linear nature of the long tails in this log-log plot directly reveals the phenomenon of heavy-tails.

The formal explanation for heavy-tailed behavior comes from the fact that there is a

²Note that the ratio of pre-assigned cells corresponds to the complement of the ratio of holes, *i.e.*, $1 - h/n^2$.

³The exact location of the phase transition appears to be characterized in terms of $(1-h)/n^p$, where p is around 1.55. However, and given that for low orders of quasigroups $p = 2$ is a good approximation, for simplification we talk about proportion of preassigned colors in terms of the total number of cells of the matrix, *i.e.*, N^2 [1].

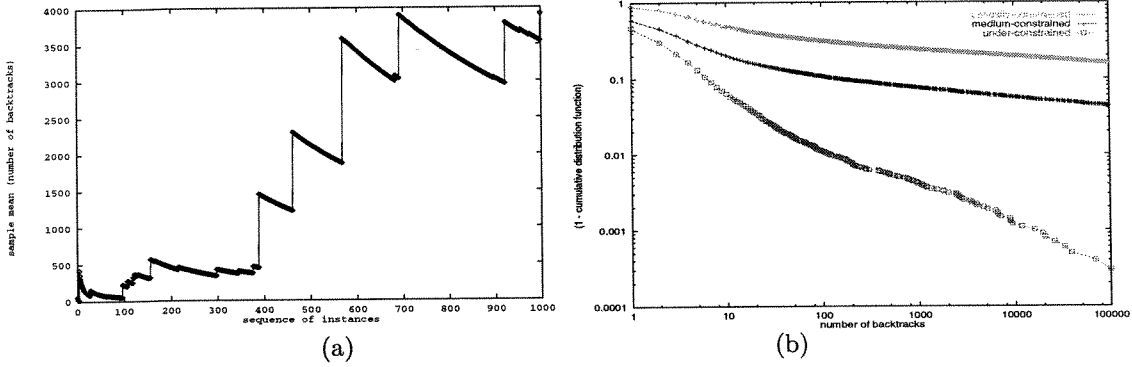


Figure 3: (a) Erratic behavior of mean cost value. (b) Log-log plot of heavy-tailed behavior.

non zero probability of entering a subtree of exponential size that has no solutions [3]. The phenomenon of heavy-tailed distributions suggests that a sequence of “short” runs instead of a single long run may be a more effective use of our computational resources. As a direct practical consequence of the heavy-tailed behavior of cost distributions, randomized *restarts* of search procedures can dramatically reduce the variance in the search behavior. In fact, restarts eliminate heavy-tail behavior [10].

The structure implicit in QCP is similar to that found in real-world domains: indeed, many problems in scheduling and experimental design have a structure similar to the structure of QCP. A particularly interesting application that directly maps onto the QCP is the problem of assigning wavelengths to routes in fiber-optic networks, as performed by Latin routers [17]. As the name suggests, Latin routers use the concept of Latin Squares to capture the constraints required to achieve conflict-free routing: a given wavelength cannot be assigned to a given input port more than once; a given wavelength cannot be assigned to a given output port more than once.

3 Problem Formulations

3.1 CSP Formulation

Given a partial latin square of order n , PLS , the latin square completion problem can be expressed as a CSP [9]:

$$\begin{aligned}
 & x_{i,j} \in \{1, \dots, n\} \quad \forall i, j \\
 & x_{i,j} = k \quad \forall i, j \text{ such that } PLS_{ij} = k \\
 & \text{alldiff } (x_{i,1}, x_{i,2}, \dots, x_{i,n}) \quad \forall i = 1, 2, \dots, n \\
 & \text{alldiff } (x_{1,j}, x_{2,j}, \dots, x_{n,j}) \quad \forall j = 1, 2, \dots, n
 \end{aligned}$$

The alldiff constraint states that all the variables involved in the constraint have to have different values. It has been shown that a CSP approach solves QCP instances up

to order around 33 relatively well [9, 22, 1]. However, for higher orders, instances in the critically constrained area are beyond the reach of pure CSP solvers, given the highly exponential behavior in this region.

3.2 Assignment Formulation

Given a partial latin square of order n , PLS , the latin square completion problem can be expressed as an integer program [17]:

$$\begin{aligned}
& \max \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n x_{i,j,k} \\
& \text{subject to} \\
& \sum_{i=1}^n x_{i,j,k} \leq 1, \quad \forall j, k \\
& \sum_{j=1}^n x_{i,j,k} \leq 1, \quad \forall i, k \\
& \sum_{k=1}^n x_{i,j,k} \leq 1, \quad \forall i, j \\
& x_{i,j,k} = 1 \quad \forall i, j, k \text{ such that } PLS_{ij} = k \\
& x_{i,j,k} \in \{0, 1\} \quad \forall i, j, k \\
& i, j, k = 1, 2, \dots, n
\end{aligned}$$

If PLS is completable, the optimal value of this integer program is h , *i.e.*, the number of holes in PLS . Kumar et al. [17] considered the design of approximation algorithms for this optimization variant of the problem based on first solving the linear programming relaxation of this integer programming formulation; that is, the conditions $x_{i,j,k} \in \{0, 1\}$ above are replaced by $x_{i,j,k} \geq 0$. Their algorithm repeatedly solves this linear programming relaxation, focuses on the variable closest to 1 (among those not set to 1 by the PLS conditions), and sets that variable to 1; this iterates until all variables are set. This algorithm is shown to be a $1/3$ -approximation algorithm; that is, if PLS is completable, then it manages to find an extension that fills at least $h/3$ holes. Kumar et al. also provide a more sophisticated algorithm in which the colors are considered in turn; in the iteration corresponding to color k , the algorithm finds the extension (of at most n cells) for which the linear programming relaxation places the greatest total weight. This algorithm is shown to be a $1/2$ -approximation algorithm; that is, if PLS is completable, then the algorithm computes an extension that fills at least $h/2$ holes. In the experimental evaluation of their algorithms, Kumar et al. solve problems up to order 9.

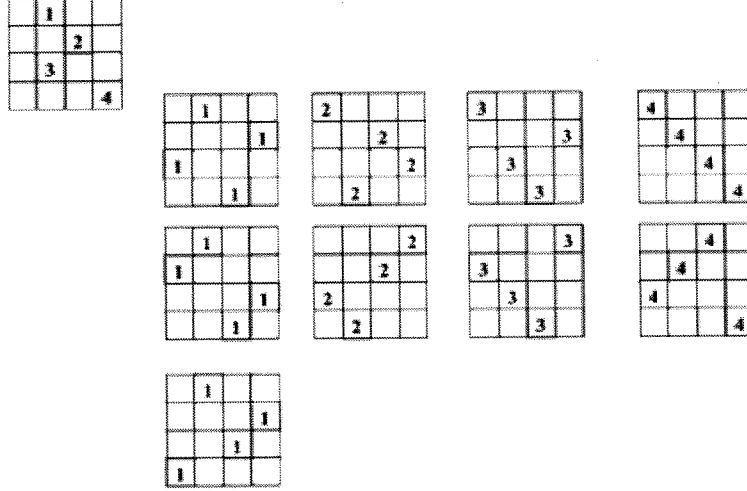


Figure 4: Families of compatible matchings for the partial latin square in the left upper corner. For example, the family of compatible matchings for symbol 1 has three compatible matchings.

3.3 Packing Formulation

Alternate integer programming formulations of this problem can also be considered. The *packing formulation* is one such formulation for which the linear programming relaxation produces stronger lower bounds. For the given PLS input, consider one color k . If PLS is completable, then there must be an extension of this solution with respect to this one color; that is, there is a set of cells (i, j) that can each be colored k so that there is exactly one cell colored k in every row and column. We shall call one such collection of cells a *compatible matching* for k . Furthermore, any subset of a compatible matching shall be called a *compatible partial matching*; let \mathcal{M}_k denote the family of all compatible partial matchings.

With this notation in mind, then we can generate the following integer programming formulation by introducing one variable $y_{k,M}$ for each compatible partial matching M in \mathcal{M}_k :

$$\begin{aligned}
 & \max \sum_{k=1}^n \sum_{M \in \mathcal{M}_k} |M| y_{k,M} \\
 & \text{subject to} \\
 & \sum_{M \in \mathcal{M}_k} y_{k,M} = 1, \quad \forall k \\
 & \sum_{k=1}^n \sum_{M \in \mathcal{M}_k: (i,j) \in M} y_{k,M} \leq 1, \quad \forall i, j \\
 & y_{k,M} \in \{0, 1\} \quad \forall k, M.
 \end{aligned}$$

Once again, we can consider the linear programming relaxation of this formulation, in which the binary constraints are relaxed to be merely non-negativity constraints. It is significant to note that, for any feasible solution y to this linear programming relaxation, one can generate a corresponding feasible solution x to the assignment formulation, by simply computing $x_{i,j,k} = \sum_{M \in \mathcal{M}_k} y_{k,M}$. This construction implies that the value of the linear programming relaxation of the assignment formulation (which provides an upper bound on the desired integer programming formulation) is at least the bound implied by the LP relaxation of the packing formulation; that is, the packing formulation provides a tighter upper bound. However, note that the size of this formulation is exponential in n . In spite of this difficulty, one may apply column generation techniques (see, e.g., the textbook by [5]) to compute an optimal solution relatively efficiently.

4 Approximations Based on Randomized Rounding

One important area of recent research has been the design of approximation algorithms in which good solutions are computed for an integer programming problem in which the variables are constrained to be 0 or 1 by solving its linear programming relaxation, and (appropriately) interpreting the resulting fractional solution as providing a probability distribution over which to set the variables to 1.

Consider the generic integer program $\max cz$ subject to $Az = b$, $z \in \{0, 1\}^N$, and solve its linear relaxation to obtain z^* . If each variable z_j is then set to 1 with probability z_j^* , then the expected value of the resulting integer solution is equal to LP optimal value, and, for each constraint, the expected value of the left-hand side is equal to the right-hand side. Of course, this does not mean that the resulting solution is feasible, but it provides a powerful intuition for why such a *randomized rounding* is a useful algorithmic tool.

This approach has led to striking results in a number of settings. For example, Goemans and Williamson [7] have given a 3/4-approximation algorithm based on randomized rounding for the problem of satisfying the maximum number of clauses for a boolean formula in conjunctive normal form. This algorithm outputs the better solution found by two randomized rounding procedures, one that uses a fair coin to independently set the variables, and another that randomly rounds based on the optimal solution to a natural linear programming relaxation.

4.1 Assignment Formulation

The assignment formulation can be used as the basis for a randomized rounding procedure in a variety of ways. Let x^* denote an optimal solution to the linear programming relaxation of this integer program. For any randomized procedure in which the probability that cell (i, j) is colored k is equal to x_{ijk}^* , then we know that, in expectation, each row i has at most one element of each color k , each column j has at most one element of each color k , and each cell (i, j) is assigned at most one color k . However, having these each hold “in expectation” is quite different than expecting that all of them will hold simultaneously, which is extremely unlikely.

4.2 Packing Formulation

In contrast to the situation for the assignment formulation, there is an easy theoretical justification for the randomized rounding of the fractional optimal solution, as we proposed in [21]. Rather than the generic randomized rounding mentioned above, instead, for each color k choose some compatible partial matching M with probability $y_{k,M}$ (so that some matching is therefore selected for each color). These selections are done as independent random events. This independence implies there might be some cell (i, j) included in the matching selected for two distinct colors. However, the constraints in the linear program imply that the expected number of matchings in which a cell is included is at most one. In fact, if PLS is completable, and hence the linear programming relaxation satisfies the inequality constraints with equality (and hence $|M| = n$ whenever $y_{k,M} > 0$), then it is straightforward to show that the expected number of cells for which some such conflict exists is at most h/e ; that is, at least $(1 - 1/e)h$ holes can be expected to be filled by this technique.

5 Hybrid CSP/LP Randomized Rounding Backtrack Search

We now describe a complete randomized backtrack search algorithm for the quasigroup (latin square) completion problem.

A central feature of the algorithm is the fact that it maintains two different formulations of the quasigroup completion problem: the CSP formulation, as described in section 3.1, and the relaxation of the LP formulation described in section 3.2.⁴ The hybrid nature of the algorithm results from the combination of strategies for variable and value assignment, and propagation, based on the two underlying models.

The algorithm is initialized by populating the CSP model and propagating constraints over this model. The CSP model is implemented in Ilog/Solver [14]. For the propagation of the ALLDIFF constraint we use the extended version provided by Ilog [14, 20]. The updated domain values are then used to populate the LP model. We solve the LP model using Ilog/Cplex Barrier [13].

As we will see from our experiments below, the LP provides valuable search guidance and pruning information for the CSP search. However, since solving the LP model is relatively expensive compared to the inference steps in the CSP model, we have to carefully manage the time spent on solving the LP model. The LP effort is controlled by two parameters, as explained below.

At the top of the backtrack search tree, variable and value selection are based on the LP rounding. After each value assignment based on the LP, full propagation is performed on the CSP model. The percentage of variables set by the LP is controlled by the parameter %LP. (So, with % LP = 0, we have a pure CSP strategy.) After this initial phase, variable and value settings are based purely on the CSP model. Note that deeper down in the search tree, the LP formulation continues to provide information on

⁴In the experiments reported here, we are using the assignment formulation for the LP. Even though the packing formulation has stronger theoretical bounds, because we solve this formulation using column generation, it is more difficult to show a concrete payoff of this approach, in practice. We are pursuing further experiments with this approach.

variable settings. However, we have found that this information can be computed more efficiently through the CSP model.

Ideally, in order to increase the accuracy of the variable assignments based on LP-rounding, one would like to update and re-solve the LP model after each variable setting. However, in practice, this is too expensive. We therefore introduce a parameter, *interleave-LP*, which determines the frequency with which the LP model is updated and re-solved. In our experiments, we found that updating the LP model after every five variable settings (*interleave-LP* = 5) is a good compromise.

In our LP rounding strategy, we first rank the variables according to their LP values (*i.e.*, variables with LP values closest to 1 are ranked near the top). We then select the highest ranked variable and set its value to 1 (*i.e.*, set the color of the corresponding cell) with a probability p given by its LP value. With probability $1 - p$, we randomly select a color for the cell from the colors still allowed according to the CSP model. After each variable setting, we perform CSP propagation. The CSP propagation will set some of the variables on our ranked variable list. We then consider the next highest ranked variable that is not yet assigned. A total of *interleave-LP* variables is assigned this way, before we update and re-solve the LP. In the search guided by the CSP model, we use a variant of the Brelaz heuristic [2, 9] for the variable and value selections.

Backtracking can occur as a result of an inconsistency detected either by the CSP model or the LP relaxation. It is interesting to note that backtracking based on inconsistencies detected by the LP occurs rather frequently at the top of our search tree. This means that the LP does indeed uncover global information not easily obtained via CSP propagation, which is a more local inference process. Of course, as noted before, lower down in the search tree, using the LP for pruning becomes ineffective since CSP propagation with only a few additional backtracks can uncover the same information.

In this setting, we are effectively using the LP values as heuristic guidance, using a randomized rounding approach inspired by the rounding schemes used in approximation algorithms. As we discussed in section 3.3, for the packing formulation of the LP, we have a clear theoretical basis for such a rounding scheme. For the assignment formulation, the theoretical justification is less immediate — nevertheless, as we will see below, our rounding scheme leads to a clear practical payoff.

Finally, we use a cutoff parameter to control our backtrack search. As mentioned in section 2, backtrack search methods are characterized by heavy-tailed behavior. That is, a backtrack search is quite likely to encounter extremely long runs. To avoid getting stuck in such unproductive runs, we use a cutoff parameter. This parameter defines the number of backtracks after which the search is restarted, at the beginning of the search tree, with a different random seed. Note that in order to maintain the completeness of the algorithm we just have to increase the cutoff. In the limit, we run the algorithm without a cutoff.

6 Empirical Results

To investigate our hypothesis that the LP relaxation can provide useful search guidance, we focused our empirical evaluation on solvable instances. To do so, we used a variant of the QCP problem, in which we generate instances for which we are guaranteed that a solution exists. To obtain such instances, we start with a randomly generated complete latin square and uncolor a fraction of cells (randomly selected). The random complete

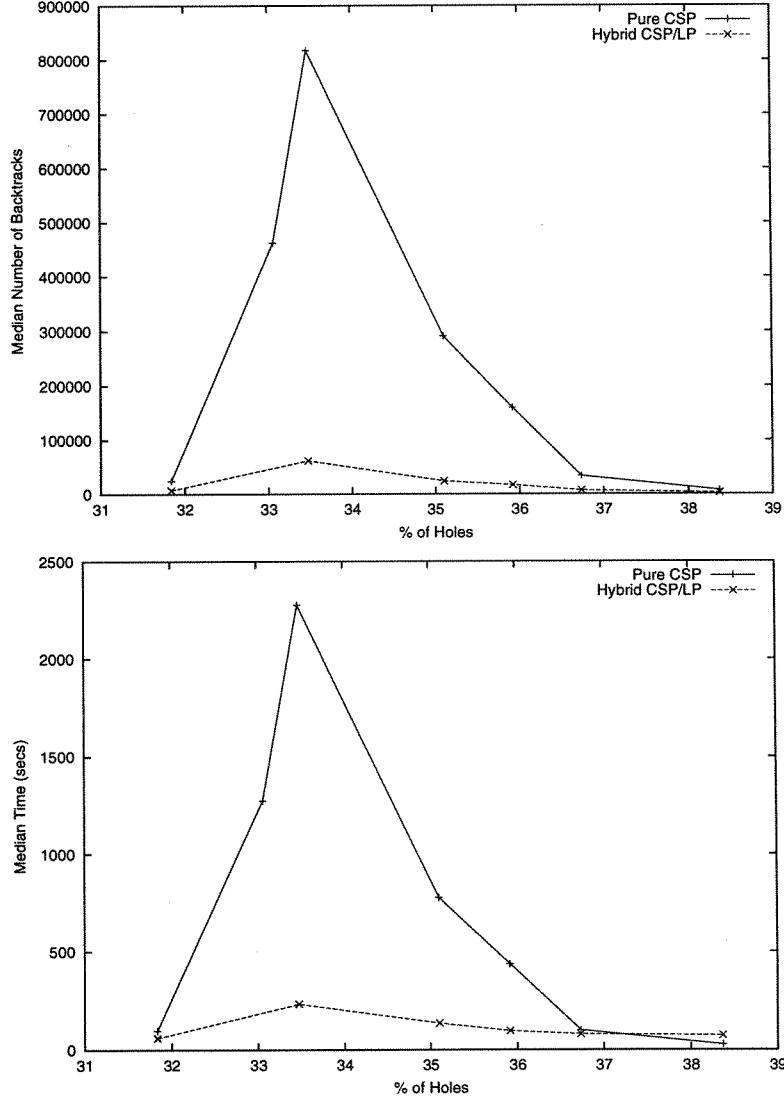


Figure 5: Median run time (secs) for QWH instances of order 35. (100 instances per data point)

latin square is generated using a Markov chain Monte Carlo shuffling process [15]. The task again is to find a coloring for the empty cells that completes the Latin square. We refer to this problem as the “quasigroup with holes” (QWH) problem.⁵ We can again finely tune the complexity of the completion task by varying the fraction of the uncolored cells [1].

⁵The code for this generator is available by contacting Carla Gomes (gomes@cs.cornell.edu).

In Figure 5, we compare the performance of our hybrid CSP/LP strategy against the pure CSP strategy. For the hybrid CSP/LP strategy we set $\%LP = 10$ and $interleave-LP = 5$. Each data point was obtained by running the randomized search procedures on 100 different instances, with a cutoff of 10^6 , and computing the median in number of backtracks (upper panel) and total run time (lower panel). From the figure, we again see the easy-hard-easy pattern, both in the hybrid CSP/LP and the pure CSP strategy. Moreover, the hybrid CSP/LP strategy significantly outperforms the pure CSP strategy, both in terms of the number of backtracks and total run time. The relative payoff of our hybrid strategy is largest for the hardest problem instances (around 33.6% holes).

We now consider more detailed performance data on three hard problem instances. In Table 1 we show the performance of the CSP/LP strategy and the pure CSP strategy on an instance of order 35 with 405 holes (33% holes). (This instance is medium hard — somewhat before the phase transition region.) The pure CSP strategy can solve this instance using a high cutoff of 10^6 , but only in 6% of the runs. On the other hand, the CSP/LP strategy is much more effective. In fact, even with only $\%LP = 1$, we can solve the instance in 42% of the runs. With $\%LP \geq 10$, we solve the instance on each run. Looking at the overall run time as a function of $\%LP$, we see that at some point further use of LP relaxations becomes counterproductive. The best performance is obtained with $\%LP$ around 20.

$\%LP$	Cutoff	Num. Runs	% Succ. Runs	Median Backtracks	Median Time
0	10^6	100	6%	474049	1312.58
1	10^6	100	42%	589438	1992.08
5	10^6	100	90%	188582	615.16
10	10^6	100	100%	26209	114.35
15	10^6	100	100%	22615	116.29
20	10^6	100	100%	17203	112.64
25	10^6	100	100%	21489	158.07
30	10^6	100	100%	24139	179.37
50	10^6	100	100%	19325	262.67
75	10^6	100	100%	17458	379.68

Table 1: Hybrid CSP/LP search on an instance of order 35 with 405 holes.

In Table 2 and Table 3, we consider, respectively, a critically constrained instance of QWH of order 40, with 528 holes, and a medium constrained instance of QWH of order 40, with 544 holes. We were unable to solve these instances with a pure CSP strategy using a cutoff of 10^5 (100 runs) and a cutoff of 10^6 (100 runs). Both instances can be solved with the hybrid CSP/LP strategy. The success rate increases with a higher $\%LP$. From the median overall run time, we see that the best performance is obtained for $\%LP$ around 25.

We have not found any instance that could be solved with the pure CSP strategy and could not be solved with the CSP/LP strategy. Furthermore, hard instances of QCP/QWH appear out of reach of a pure integer programming strategy (i.e., no interleaved CSP propagation): The pruning power provided by the CSP component is critical in this highly combinatorial domain.

% LP	Cutoff	Num. Runs	% Succ. Runs	Median Backtracks	Median Time
0	10^5	100	0%	N.A.	N.A.
10	10^5	100	1%	48387	245.54
25	10^5	100	37%	17382	215.69
50	10^5	100	47%	21643	422.59
0	10^6	100	0%	N.A.	N.A.
10	10^6	100	8%	355362	1488.08
25	10^6	100	64%	123739	574.68
50	10^6	100	65.3%	128306	757.55

Table 2: Instance of order 40, with 528 holes.

% LP	Cutoff	Num. Runs	% Succ. Runs	Median Backtracks	Median Time
0	10^5	100	0%	N.A.	N.A.
10	10^5	100	1%	41771	264.96
25	10^5	100	34%	31386	287.72
50	10^5	100	38%	13266	395.31
0	10^6	100	0%	N.A.	N.A.
10	10^6	100	5%	167897	813.58
25	10^6	100	53%	110787	560.56
50	10^6	100	92%	75234	648.87

Table 3: Instance of order 40, with 544 holes.

Overall, our hybrid method significantly extends the range of QWH problems we can solve. The hybrid strategy allows us to improve on the time performance of the pure CSP strategy and reliably solve larger instances, up to order 40 to 45. We also solved several hard instances of order 50. On our very hardest problem, we had to increase %LP to around 50%. So, apparently, more guidance was required from the LP relaxation.

7 Conclusions

Constraint based search techniques have shown to be remarkably efficient on purely combinatorial problems. In many domains, such techniques outperform integer programming based approaches. Nevertheless, LP relaxations may still provide useful information, for example, in guiding constraint based search techniques.

We have demonstrated the promise of boosting CSP methods using LP relaxations on hard, purely combinatorial problems. Our approach involves a randomized rounding strategy inspired by recent rounding methods used in approximation algorithms. In this setting, the LP provides powerful guidance in the CSP search. Randomization and restarts in the backtrack process are needed to make the overall strategy robust and to recover from possible early branching mistakes.

Essential to our approach is a tight coupling between the CSP and LP method. We

simultaneously maintain a CSP and an LP model of the problem. The local nature and high efficiency of the CSP propagation methods enable us to call such methods frequently. In particular, CSP propagation is performed after each variable assignment. By frequently updating and resolving the LP model, our LP rounding decisions stay accurate during the search process. The continuous interleaving of CSP propagation and LP heuristic guidance using randomized rounding are key features of our approach. Also, we carefully control the amount of time spent in solving the LP relaxations, by restricting this process to the top half of the backtrack search tree and not solving the LP at every node of the search tree.

In experiments, we were able to significantly extend the reach of CSP and LP techniques for solving instances of the quasigroup completion problem. Our technique is general and therefore holds promise for a range of combinatorial problems.

We believe there is still room for further improvement. For example, using different CSP and LP formulations and different rounding strategies. In particular, we are currently experimenting with the packing formulation which offers better theoretical guarantees.

References

- [1] D. Achlioptas, Carla Gomes, Henry Kautz, and Bart Selman. Generating Satisfiable Instances. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-00)*, New Providence, RI, 2000. AAAI Press.
- [2] D. Brelaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.
- [3] H. Chen, C. Gomes, and B. Selman. Formal models of heavy-tailed behavior in combinatorial search. In *Proceedings of 7th Intl. Conference on the Principles and Practice of Constraint Programming (CP-2001)*, *Lecture Notes in Computer Science*, Vol. 2239, Springer-Verlag, pages 408–422, 2001.
- [4] F. Chudak and D. Shmoys. Improved approximation algorithms for the uncapacitated facility location problem. In *Submitted for publication*, 1999. Preliminary version of this paper (with the same title) appeared in proceedings of the Sixth Conference on Integer Programming and Combinatorial Optimization.
- [5] Vasek Chvatal. *Linear Programming*. W.H.Freeman Company, 1983.
- [6] C. Colbourn. The complexity of completing partial latin squares. *Discrete Applied Mathematics*, (8):25–30, 1984.
- [7] M. X. Goemans and D. P. Williamson. 0.878-approximation algorithms for max-cut and max-sat. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Theory of Computing*, pages 422–431, 1994.
- [8] Carla Gomes. editor, *The Knowledge Engineering Review. Special Issue on the Integration of Artificial Intelligence and Operations Research Techniques*, Vol. 15 (1) and Vol. 16 (1). Cambridge Press, 2000-2001.

- [9] Carla Gomes and Bart Selman. Problem Structure in the Presence of Perturbations. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 221–227, New Providence, RI, 1997. AAAI Press.
- [10] Carla Gomes, Bart Selman, Nuno Crato, and Henry Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. of Automated Reasoning*, 24(1–2):67–100, 2000.
- [11] John Hooker. Resolution vs. cutting plane solution of inference problems: Some computational experience. *Operations Research Letter*, 7(1):1–7, 1988.
- [12] John Hooker. Resolution and the integrality of satisfiability problems. *Mathematical Programming*, 74:1–10, 1996.
- [13] Ilog Inc. Ilog cplex 7.1. user’s manual., 2001.
- [14] Ilog Inc. Ilog solver 5.1. user’s manual., 2001.
- [15] M.T. Jacobson and P. Matthews. Generating uniformly distributed random latin squares. *J. of Combinatorial Designs*, 4(6):405–437, 1996.
- [16] A.P. Kamath, N. K. Karmarkar, K. G. Ramakrishnan, and M. G. C. Resende. A continuous approach to inductive inference. *Mathematical Programming*, 57:215–238, 1992.
- [17] S. R. Kumar, A. Russell, and R. Sundaram. Approximating latin square extensions. *Algorithmica*, 24:128–138, 1999.
- [18] Charles Laywine and Gary Mullen. *Discrete Mathematics using Latin Squares*. Wiley-Interscience Series in Discrete mathematics and Optimization, 1998.
- [19] Rajeev Motwani, Joseph Naor, and Prabhakar Raghavan. Randomized approximation algorithms in combinatorial optimization. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, 1997.
- [20] J. C. Regin. A filtering algorithm for constraints of difference in csp. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 362–367, Seattle, WA, 1994. AAAI Press.
- [21] R. Regis, C. Gomes, and D. Shmoys. An improvement performance guarantee for the partial latin square problem. Manuscript in preparation, 2002.
- [22] P. Shaw, K. Stergiou, and T. Walsh. Arc consistency and quasigroup completion. In *Proceedings of ECAI-98, workshop on binary constraints*, 1998.
- [23] Johannes Warners. *Nonlinear approaches to satisfiability problems*. PhD thesis, Technische Universiteit Eindhoven, 1999.

Appendix 2

Formal Models of Heavy-Tailed Behavior in Combinatorial Search^{*}

Hubie Chen, Carla Gomes, and Bart Selman

Department of Computer Science, Cornell University, Ithaca, NY 14853, USA
{hubes,gomes,selman}@cs.cornell.edu

Abstract. Recently, it has been found that the cost distributions of randomized backtrack search in combinatorial domains are often heavy-tailed. Such heavy-tailed distributions explain the high variability observed when using backtrack-style procedures. A good understanding of this phenomenon can lead to better search techniques. For example, restart strategies provide a good mechanism for eliminating the heavy-tailed behavior and boosting the overall search performance. Several state-of-the-art SAT solvers now incorporate such restart mechanisms. The study of heavy-tailed phenomena in combinatorial search has so far been largely based on empirical data. We introduce several abstract tree search models, and show formally how heavy-tailed cost distribution can arise in backtrack search. We also discuss how these insights may facilitate the development of better combinatorial search methods.

1 Introduction

Recently there have been a series of new insights into the high variability observed in the run time of backtrack search procedures. Empirical work has shown that the run time distributions of backtrack style algorithms often exhibit so-called *heavy-tailed* behavior [5]. Heavy-tailed probability distributions are highly non-standard distributions that capture unusually erratic behavior and large variations in random phenomena. The understanding of such phenomena in backtrack search has provided new insights into the design of search algorithms and led to new search strategies, in particular, restart strategies. Such strategies avoid the long tails in the run time distributions and take advantage of the probability mass at the beginning of the distributions. Randomization and restart strategies are now an integral part of several state-of-the-art SAT solvers, for example, Chaff [12], GRASP [11], Relsat [1], and Satz-rand [9, 4].

Research on heavy-tailed distributions and restart strategies in combinatorial search has been largely based on empirical studies of run time distributions. However, so far, a detailed rigorous understanding of such phenomena has been

^{*} This research was partially funded by AFRL, grants F30602-99-1-0005 and F30602-99-1-0006, AFOSR, grant F49620-01-1-0076 (HSI) and F49620-01-1-0361 and DARPA, F30602-00-2-0530 and F30602-00-2-0558. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government.

lacking. In this paper, we provide a formal characterization of several tree search models and show under what conditions heavy-tailed distributions can arise.

Intuitively, heavy-tailed behavior in backtrack style search arises from the fact that wrong branching decisions may lead the procedure to explore an exponentially large subtree of the search space that contains no solutions. Depending on the number of such “bad” branching choices, one can expect a large variability in the time to find a solution on different runs. Our analysis will make this intuition precise by providing a search tree model, for which we can formally prove that the run time distribution is heavy-tailed. A key component of our model is that it allows for highly irregular and imbalanced trees, which produce search times that differ radically from run to run. We also analyze a tree search model that leads to fully balanced search trees. The balanced tree model does not exhibit heavy-tailed behavior, and restart strategies are provably ineffective in this model. The contrast between the balanced and imbalanced models shows that heavy-tailedness is not inherent to backtrack search in general but rather emerges from backtrack searches through highly irregular search spaces.

Whether search trees encountered in practice correspond more closely to balanced or imbalanced trees is determined by the combination of the characteristics of the underlying problem instance and the search heuristics, pruning, and propagation methods employed. Balanced trees occur when such techniques are relatively ineffective in the problem domain under consideration. For example, certain problem instances, such as the parity formulas [2], are specifically designed to “fool” any clever search technique. (The parity problems were derived using ideas from cryptography.) On such problem instances backtrack search tends to degrade to a form of exhaustive search, and backtrack search trees correspond to nearly fully balanced trees with a depth equal to the number of independent variables in the problem. In this case, our balanced search tree model captures the statistical properties of such search spaces.

Fortunately, most CSP or SAT problems from real-world applications have much more structure, and branching heuristics, dynamic variable ordering, and pruning techniques can be quite effective. When observing backtrack search on such instances, one often observes highly imbalanced search trees. That is, there can be very short subtrees, where the heuristics (combined with propagation) quickly discover contradictions; or, at other times, the search procedure branches deeply into large subtrees, making relatively little progress in exploring the overall search space. As a result, the overall search tree becomes highly irregular, and, as our imbalanced search tree model shows, exhibits heavy-tailed behavior, often making random restarts effective.

Before proceeding with the technical details of our analysis, we now give a brief summary of our main technical results. For our balanced model, we will show that the expected run time (measured in leaf nodes visited) scales exponentially in the height of the search tree, which corresponds to the number of independent variables in the problem instance. The underlying run time distribution is not heavy-tailed, and a restart strategy will not improve the search performance.

For our imbalanced search tree model, we will show that the run time of a randomized backtrack search method is heavy-tailed, for a range of values of the model parameter p , which characterizes the effectiveness of the branching heuristics and pruning techniques. The heavy-tailedness leads to an infinite variance and sometimes an infinite mean of the run time. In this model, a restart strategy will lead to a polynomial mean and a polynomial variance.

We subsequently refine our imbalanced model by taking into account that in general we are dealing with finite-size search trees of size at most b^n , where b is the branching factor. As an immediate consequence, the run time distribution of a backtrack search is bounded and therefore cannot, strictly speaking, be heavy-tailed (which requires infinitely long “fat” tails). Our analysis shows, however, that a so-called “bounded heavy-tailed” model provides a good framework for studying the search behavior on such trees. The bounded distributions share many properties with true heavy-tailed distributions. We will show how the model gives rise to searches whose mean scales exponentially. Nevertheless, short runs have sufficient probability mass to allow for an effective restart strategy, with a mean run time that scales polynomially. These results closely mimic the properties of empirically determined run time distributions on certain classes of structured instances, and explain the practical effectiveness of restarts, as well as the large observed variability between different backtrack runs.

The key components that lead to heavy-tailed behavior in backtrack search are (1) an exponential search space and (2) effective branching heuristics with propagation mechanisms. The second criteria is necessary to create a reasonable probability mass for finding a solution early on in the search. Interestingly, our analysis suggests that heuristics that create a large variability between runs may be more effective than more uniform heuristics because a restart strategy can take advantage of some of the short, but possibly relatively rare, runs.¹

We should stress that although our imbalanced tree model results in heavy-tailed behavior, we do not mean to suggest that this is the only such model that would do so. In fact, our imbalanced model is just one possible search tree model, and it is a topic for future research to explore other search models that may also result in heavy-tailed behavior.

The paper is structured as follows. In section 2, we present our balanced tree model. In section 3, we introduce the imbalanced search tree model, followed by the bounded version in section 4. Section 5 gives the conclusions and discusses directions for future work.

2 Balanced trees

We first consider the case of a backtrack search on a balanced tree. To obtain the base-case for our analysis, we consider the most basic form of backtrack search.

¹ In an interesting study, Chu Min Li (1999) [8] argues that asymmetric heuristics may indeed be quite powerful. The study shows that heuristics that lead to “skinny” but deep search trees can be more effective than heuristics that uniformly try to minimize the overall depth of the trees, thereby creating relative short but dense trees.

We will subsequently relax our assumptions and move on to more practical forms of backtrack search. In our base model, we assume chronological backtracking, fixed variable ordering, and random child selection with no propagation or pruning. We consider a branching factor of two, although the analysis easily extends to any constant branching factor.

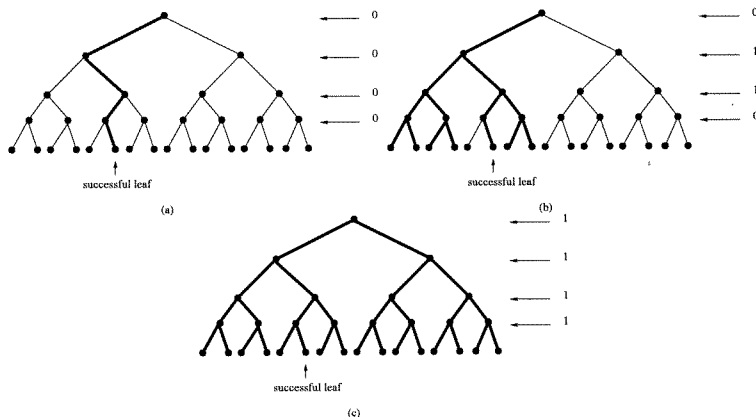


Fig. 1. Balanced tree model.

Figure 1 shows three examples of our basic setup. The full search space is a complete binary tree of depth n with 2^n leaf nodes at the bottom. We assume that there is exactly a single successful leaf.² The bold-faced subtrees show the nodes visited before the successful leaf is found. The figure is still only an abstraction of the actual search process: there are still different ways to traverse the bold-faced subtrees, referred to as “abstract search subtrees”. An abstract search tree corresponds to the tree of all visited nodes, without specification of the order in which the nodes are visited. Two different runs of a backtrack search can have the same abstract tree but different concrete search trees in which the same nodes are visited but in different order.

2.1 Probabilistic characterization of the balanced tree model

Our balanced tree search model has a number of interesting properties. For example, each abstract search subtree is characterized by a unique number of visited leaf nodes, ranging from 1 to 2^n . Moreover, once the successful leaf is fixed, each abstract subtree occurs with probability $(1/2)^n$. The number of leaf nodes visited up to and including the successful leaf node is a discrete uniformly distributed random variable: denoting this random variable by $T(n)$, we have $P[T(n) = i] = (1/2)^n$, when $i = 1, \dots, 2^n$.

² Having multiple solutions does not qualitatively change our results. In the full version of this paper, we will discuss this issue in more detail.

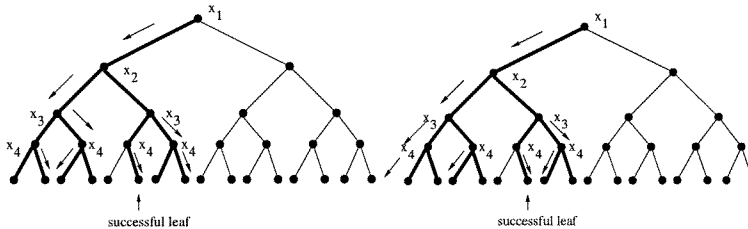


Fig. 2. Balanced tree model (detailed view).

As noted above, several runs of a backtrack search method can yield the same abstract tree, because the runs may visit the same set of nodes, but in a different order. It is useful, to also consider such actual traversals (or searches) of an abstract subtree. See Figure 2. The figure shows two possible traversals for the subtree from Figure 1(b). At each node, the figure gives the name of the branching variable selected at the node, and the arrow indicates the first visited child. The only possible variation in our search model is the order in which the children of a node are visited. To obtain the bold-faced subtree in Figure 1(b), we see that, at the top two nodes, we first need to branch to the left. Then we reach a complete subtree below node x_3 , where we have a total of 4 possible ways of traversing the subtree. In total, we have 6 possible searches that correspond to the abstract subtree in Figure 1(b).

Note that the abstract subtree in Figures 1(a) has only one possible corresponding traversal. Each possible traversal of a abstract search tree is equally likely. Therefore, the probability of an actual search traversal is given by $(1/2)^n(1/K)$, where K is the number of distinct traversals of the corresponding abstract subtree.

We now give a brief derivation of the properties of our balanced tree search. Consider the abstract binary search trees in Figure 1. Let “good” nodes be those which are ancestors of the satisfying leaf, and let “bad” nodes be all others. Our backtrack search starts at the root node; with probability $1/2$, it descends to the “bad” node at depth one, and incurs time 2^{n-1} exploring all leaves below this “bad” node. After all of these leaves have been explored, a random choice will take place at the “good” node of depth one. At this node, there is again probability $1/2$ of descending to a “good” node, and probability $1/2$ of descending to a “bad” node; in the latter case, all 2^{n-2} leaves below the “bad” node will be explored. If we continue to reason in this manner, we see that the cost of the search is

$$T(n) = X_1 2^{n-1} + \dots + X_j 2^{n-j} + \dots + X_{n-1} 2^1 + X_n 2^0 + 1$$

where each X_j is an indicator random variable, taking on the value 1 if the “bad” node at depth j was selected, and the value 0 otherwise. For each $i = 1, \dots, 2^n$, there is exactly one choice of zero-one assignments to the variables X_j so that i is equal to the above cost expression; any such assignment has probability 2^{-n} of occurring, and so this is the probability that the cost is i .

Stated differently, once the satisfying leaf is fixed, the abstract subtree is determined completely by the random variables X_j : all descendants of the “bad” sibling of the unique “good” node at depth j are explored if and only if $X_j = 1$. In Figure 1, we give the X_j settings alongside each tree. A good choice at a level gets label “0” and a bad choice gets label “1”. Each possible binary setting uniquely defines an abstract search tree and its number of leaf nodes. Hence, there are 2^n abstract subtrees, each occurring with probability $1/2^n$. The overall search cost distribution is therefore the uniform distribution over the range $i = 1, \dots, 2^n$.

This allows us to calculate the expectation and variance of the search cost in terms of the number of visited leaves, denoted by $T(n)$. The expected value is given by $E[T(n)] = \sum_{i=1}^{2^n} iP[T(n) = i]$, which with $P[T(n) = i] = 2^{-n}$ gives us $E[T(n)] = (1 + 2^n)/2$.

We also have $E[T^2(n)] = \sum_{i=1}^{2^n} i^2P[T = i]$, which equals $(2^{2n+1} + 3 \cdot 2^n + 1)/(6)$. So, for the variance we obtain $\text{Var}[T] = E[T^2(n)] - E[T(n)]^2$, which equals $(2^{2n} - 1)/(12)$.

These results show that both the expected run time and the variance of chronological backtrack search on a complete balanced tree scale exponentially in n . Of course, given that we assume that the leaf is located somewhere uniformly at random on the fringe of the tree, it makes intuitive sense that the expected search time is of the order of half of the size of the fringe. However, we have given a much more detailed analysis of the search process to provide a better understanding of the full probability distribution over the search trees and abstract search trees.

2.2 The effect of restarts

We conclude our analysis of the balanced case by considering whether a randomized restart strategy can be beneficial in this setting. As discussed earlier, restart strategies for randomized backtrack search have shown to be quite effective in practice [4]. However, in the balanced search tree model, a restart strategy is not effective in reducing the run time to a polynomial.

In our analysis, we slightly relax the assumptions made about our search model. We assume a branching factor of $b \geq 2$, and we make no assumptions about the order in which the algorithm visits the children of an internal node, other than that the first child is picked randomly. Indeed, our analysis applies even if an arbitrarily intelligent heuristic is used to select among the remaining unvisited children at a node. However, for the case of $b = 2$, this model is identical to our previous model. As we will see, the mean of $T(n)$ is still exponential.

Our first observation gives the probability that the number of visited leaf nodes $T(n)$ does not exceed a power of b .

Lemma 1. *For any integers n, k such that $0 \leq k \leq n$ and $1 \leq n$, $P[T(n) \leq b^{n-k}] = b^{-k}$.*

Proof. Observe that $T(n) \leq b^{n-k}$ if and only if at least the first k guesses are correct. The probability that the first k guesses are correct is b^{-k} . \square

It follows that the expected run time is exponential, as one would expect.

Theorem 1. *The expectation of the run time, $E[T(n)]$, for a balanced tree model is exponential in n .*

Proof. By Lemma 1, $P[T(n) \leq b^{n-1}] = b^{-1}$. Thus, $E[T(n)]$ is bounded below by $b^{n-1}(1 - b^{-1})$, which is exponential in n .

We now refine Lemma 1 to obtain an upper bound on the probability that $T(n)$ is below $f(n)$.³

Lemma 2. *If $f : \mathbb{N}^+ \rightarrow \mathbb{N}^+$ is a function such that $f(n) \leq b^n$ (for all $n \geq 1$), then $P[T(n) \leq f(n)] \leq f(n)/b^{n-1}$ (for all $n \geq 1$).*

Proof. We have that $0 \leq \log_b f(n) \leq n$. Set $k(n) = n - \log_b f(n)$, so that $\log_b f(n) = n - k(n)$. Then, $0 \leq n - k(n) \leq n$, implying that $0 \leq k(n) \leq n$. Since $0 \leq \lfloor k(n) \rfloor \leq n$, we can apply Lemma 1 to $\lfloor k(n) \rfloor$ to obtain $P[T(n) \leq b^{n-\lfloor k(n) \rfloor}] = 1/b^{\lfloor k(n) \rfloor}$. So, we have $P[T(n) \leq f(n)] = P[T(n) \leq b^{\log_b f(n)}] \leq P[T(n) \leq b^{n-\lfloor k(n) \rfloor}] = 1/b^{\lfloor k(n) \rfloor} \leq 1/b^{n-\log_b f(n)-1} \leq f(n)/b^{n-1}$. \square

This theorem implies that the probability of the search terminating in polynomial time is exponentially small in n , as $f(n)/b^{n-1}$ is exponentially small in n for any polynomial f . Using this observation, we can now show that there does not exist a restart strategy that leads to expected polynomial time performance.

Formally, a restart strategy is a sequence of times $t_1(n), t_2(n), t_3(n), \dots$. Given a randomized algorithm A and a problem instance I of size n , we can run A under the restart strategy by first executing A on I for time $t_1(n)$, followed by restarting A and running for time $t_2(n)$, and so on until a solution is found. The expected time of A running under a restart strategy can be substantially different from the expected time of running A without restarts. In particular, if the run time distribution of A is “heavy-tailed”, there is a good chance of having very long runs. In this case, a restart strategy can be used to cut off the long runs and dramatically reduce the expected run time and its variance.

Luby *et al.* [10] show that optimal performance can be obtained by using a purely *uniform* restart strategy. In a uniform strategy, each restart interval is the same, *i.e.*, $t(n) = t_1(n) = t_2(n) = t_3(n) = \dots$, where $t(n)$ is the “uniform restart time”.

Theorem 2. *Backtrack search on the balanced tree model has no uniform restart strategy with expected polynomial time.*

Proof. We prove this by contradiction. Let $t(n)$ be a uniform restart time yielding expected polynomial time. Using a lemma proved in the long version of this paper, we can assume $t(n)$ to be a polynomial. If we let the algorithm run for

³ Note on notation: We let \mathbb{N}^+ denote the set of positive integers, *i.e.*, $\{1, 2, 3, \dots\}$. We say that a function $f : \mathbb{N}^+ \rightarrow \mathbb{N}^+$ is exponential if there exist constants $c > 0$ and $b > 1$ such that $f(n) > cb^n$ for all $n \in \mathbb{N}^+$.

time $t(n)$, the probability that the algorithm finds a solution is $P[T(n) \leq t(n)]$, which by Lemma 2 is bounded above by $t(n)/b^{n-1}$. Thus, the expected time of the uniform restart strategy $t(n)$ is bounded below by $t(n)[t(n)/b^{n-1}]^{-1} = b^{n-1}$, a contradiction. \square

3 The imbalanced tree model: Heavy-tails and restarts

Before we present a tree search model where a restart strategy does work, it is useful to understand intuitively why restarts do not enhance the search on a balanced tree: When we consider the cumulative run time distribution, there is simply not enough probability mass for small search trees to obtain a polynomial expected run time when using restarts. In other words, the probability of encountering a small successful search tree is too low. This is of course a direct consequence of the balanced nature of our trees, which means that in the search all branches reach down to the maximum possible depth. This means that if one follows a path down from the top, as soon as a branching decision is made that deviates from a path to a solution, say at depth i , a full subtree of depth $n - i$ needs to be explored.

Assume that in our balanced model, our branching heuristics make an error with probability p (for random branching, we have $p = 1/2$). The probability of making the first incorrect branching choice at the i^{th} level from the top is $p(1 - p)^{i-1}$. As a consequence, with probability p , we need to explore half of the full search tree, which leads directly to an exponential expected search cost. There are only two ways to fix this problem. One way would be to have very clever heuristics ($p \ll 1$) that manage to eliminate almost all branching errors and have a reasonable chance of making the first wrong choice close to the fringe of the search tree. However, it appears unlikely that such heuristics would exist for any interesting search problem. (Such heuristics in effect almost need to solve the problem.)

Another way to remedy the situation is by having a combination of non-chronological backtracking, dynamic variable ordering, pruning, propagation, clause or constraint learning, and variable selection that terminate branches early on in a “bad subtree”.⁴ Such techniques can substantially shrink the unsuccessful subtrees. (Below, we will refer to the collection of such techniques as “CSP techniques”.) The resulting search method will be allowed to make branching mistakes but the effect of those errors will not necessarily lead to subtrees exponential in the full problem size. Of course, the resulting overall search trees will be highly irregular and may vary dramatically from run to run. As noted in the introduction, such large variations between runs have been observed in practice for a range of state-of-the-art randomized backtrack search methods.

⁴ A particularly exciting recent development is the Chaff [12] SAT solver. In a variety of structured domains, such as protocol verification, Chaff substantially extends the range of solvable instances. Chaff combines a rapid restart strategy with clause learning. The learned clauses help in pruning branches and subtrees on future restarts.

The underlying distributions are often “heavy-tailed”, and in addition, restart strategies can be highly effective.

Heavy-tailed probability distributions are formally characterized by tails that have a *power-law* (polynomial) decay, *i.e.*, distributions which asymptotically have “heavy tails” — also called tails of the Pareto-Lévy form:

$$P[X > x] \sim Cx^{-\alpha}, \quad x > 0 \quad (\star)$$

where $0 < \alpha < 2$ and $C > 0$ are constants. Some of the moments of heavy-tailed distributions are infinite. In particular, if $0 < \alpha \leq 1$, the distribution has infinite mean and infinite variance; with $1 < \alpha < 2$, the mean is finite but the variance is infinite.

We now introduce an abstract probability model for the search tree size that, depending on the choice of its characteristic parameter setting, leads to heavy-tailed behavior with an effective restart strategy. Our model was inspired by the analysis of methods for sequential decoding by Jacobs and Berlekamp [7].

Our imbalanced tree model assumes that the CSP techniques lead to an overall probability of $1 - p$ of guiding the search directly to a solution.⁵ With probability $p(1 - p)$, a search space of size b , with $b \geq 2$, needs to be explored. In general, with probability $p^i(1 - p)$, a search space of b^i nodes needs to be explored. Intuitively, p provides a probability that the overall amount of backtracking increases geometrically by a factor of b . This increase in backtracking is modeled as a global phenomenon.

More formally, our generative model leads to the following distribution. Let p be a probability ($0 < p < 1$), and $b \geq 2$ be an integer. Let T be a random variable taking on the value b^i with probability $(1 - p)p^i$, for all integers $i \geq 0$. Note that for all $\sum_{i \geq 0} (1 - p)p^i = 1$ for $0 < p < 1$, so this is indeed a well-specified probability distribution.

We will see that the larger b and p are, the “heavier” the tail. Indeed, when b and p are sufficiently large, so that their product is greater than one, the expectation of T is infinite. However, if the product of b and p is below one, then the expectation of T is finite. Similarly, if the product of b^2 and p is greater than one, the variance of T is infinite, otherwise it is finite. We now state these results formally.

The expected run time can be calculated as $E[T] = \sum_{i \geq 0} P[T = b^i]b^i = \sum_{i \geq 0} (1 - p)p^i b^i = (1 - p) \sum_{i \geq 0} (pb)^i$.

Therefore, when p , the probability of the size of the search space increasing by a factor of b , is sufficiently large, that is, $p \geq 1/b$, we get an infinite expected search time: $E[T] \rightarrow \infty$. For $p < 1/b$ (“better search control”), we obtain a finite mean of $E[T] = (1 - p)/(1 - pb)$.

⁵ Of course, the probability $1 - p$ can be close to zero. Moreover, in a straightforward generalization, one can assume an additional polynomial number of backtracks, $q(n)$, before reaching a successful leaf. This generalization is given later for the bounded case.

To compute the variance of the run time, we first compute $E[T^2] = \sum_{i \geq 0} P[T = b^i](b^i)^2 = \sum_{i \geq 0} (1-p)p^i(b^i)^2 = (1-p) \sum_{i \geq 0} (pb^2)^i$.

Then, it can be derived from $\text{Var}[T] = E[T^2] - (E[T])^2$ that (1) for $p > 1/b^2$, the variance becomes infinite, and (2) for smaller values of p , $p \leq 1/b^2$, the variance is finite with $\text{Var}[T] = \frac{1-p}{1-pb^2} - (\frac{1-p}{1-pb})^2$.

Finally, we describe the asymptotics of the survival function of T .

Lemma 3. *For all integers $k \geq 0$, $P[T > b^k] = p^{k+1}$.*

Proof. We have $P[T > b^k] = \sum_{i=k+1}^{\infty} P[T = b^i] = \sum_{i=k+1}^{\infty} (1-p)p^i = (1-p)p^{k+1} \sum_{i=k+1}^{\infty} p^{i-(k+1)} = (1-p)p^{k+1} \sum_{j=0}^{\infty} p^j = p^{k+1}$. \square

Theorem 3. *Let p be fixed. For all real numbers $L \in (0, \infty)$, $P[T > L]$ is $\Theta(L^{\log_b p})$. In particular, for $L \in (0, \infty)$, $p^2 L^{\log_b p} < P[T > L] < L^{\log_b p}$.*

Proof. We prove the second statement, which implies the first. To obtain the lower bound, observe that $P[T > L] = P[T > b^{\lceil \log_b L \rceil}] \geq P[T > b^{\lceil \log_b L \rceil}] = p^{\lceil \log_b L \rceil + 1}$, where the last equality follows from Lemma 3. Moreover, $p^{\lceil \log_b L \rceil + 1} > p^{\log_b L + 2} = p^2 p^{\log_b L} = p^2 L^{\log_b p}$. We can upper bound the tail in a similar manner: $P[T > L] \leq P[T > b^{\lceil \log_b L \rceil}] = p^{\lceil \log_b L \rceil + 1} < p^{\log_b L} = L^{\log_b p}$. \square

Theorem 3 shows that our imbalanced tree search model leads to a heavy-tailed run time distribution whenever $p > 1/b^2$. For such a p , the α of equation (\star) is less than 2.

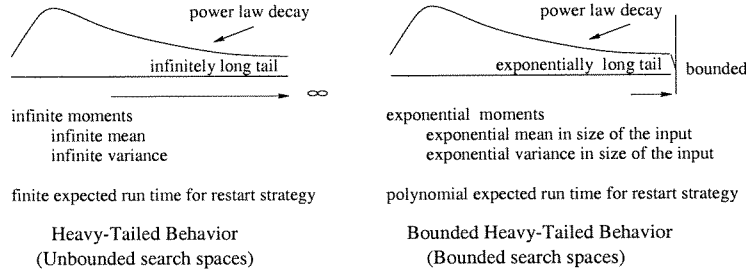


Fig. 3. Correspondence of concepts for heavy-tailed distributions and bounded heavy-tailed distributions.

4 Bounded Heavy-Tailed Behavior for Finite Distributions

Our generative model for imbalanced tree search induces a single run time distribution, and does not put an apriori bound on the size of the search space.

However, in practice, there is a different run time distribution for each combinatorial problem instance, and the run time of a backtrack search procedure on a problem instance is generally bounded above by some exponential function in the size of the instance. We can adjust our model by considering heavy-tailed distributions with bounded support or so-called “bounded heavy-tailed distributions”, for short [6]. Analogous to standard heavy-tailed distributions, the bounded version has power-law decay of the tail of the distribution (see equation (\star) over a finite, but exponential range of values. Our analysis of the bounded search space case shows that the main properties of the run time distribution observed for the unbounded imbalanced search model have natural analogues when dealing with finite but exponential size search spaces.

Figure 3 highlights the correspondence of concepts between the (unbounded) heavy-tailed model and the bounded heavy-tailed model. The key issues are: heavy-tailed distributions have infinitely long tails with power-law decay, while bounded heavy-tailed distributions have exponentially long tails with power-law decay; the concept of infinite mean in the context of a heavy-tailed distribution translates into an exponential mean in the size of the input, when considering bounded heavy-tailed distributions; a restart strategy applied to a backtrack search procedure with heavy-tailed behavior has a finite expected run time, while, in the case of bounded search spaces, we are interested in restart strategies that lead to a polynomial expected run time, whereas the original search algorithm (without restarts) exhibits bounded heavy-tailed behavior with an exponential expected run time. Furthermore, we should point out that exactly the same phenomena that lead to heavy-tailed behavior in the imbalanced generative model — the conjugation of an exponentially decreasing probability of a series of “mistakes” with an exponentially increasing penalty in the size of the space to search — cause bounded heavy-tailed behavior with an exponential mean in the bounded case.

To make this discussion more concrete, we now consider the bounded version of our imbalanced tree model. We put a bound of n on the depth of the generative model and normalize the probabilities accordingly. The run time $T(n)$ for our search model can take on values $b^i q(n)$ with probability $P[T(n) = b^i q(n)] = Cp^i$, for $i = 0, 1, 2, \dots, n$. We renormalize this distribution using a sequence of constants C_n , which is set equal to $\frac{1-p}{1-p^{n+1}}$. This guarantees that we obtain a valid probability distribution, since $\sum_{i=0}^n C_n p^i = 1$. Note that $C_n < 1$ for all $n \geq 1$. We assume $b > 1$ and that $q(n)$ is a polynomial in n .

For the expected run time we have $E[T] = \sum_{i=0}^n P[T = b^i q(n)](b^i q(n)) = \sum_{i=0}^n (C_n p^i)(b^i q(n)) = C_n q(n) \sum_{i=0}^n (pb)^i$.

We can distinguish two cases.

(1) For $p \leq 1/b$, we have $E[T] \leq C_n q(n)(n+1)$.

(2) For $p > 1/b$, we obtain a mean that is exponential in n , because we have $E[T] \geq C_n q(n)(pb)^n$.

We compute the variance as follows. First, we have $E[T^2] = \sum_{i=0}^n P[T = b^i q(n)](b^i q(n))^2 = \sum_{i=0}^n C_n p^i (b^{2i} q^2(n)) = C_n q^2(n) \sum_{i=0}^n (pb^2)^i$. From $\text{Var}[T] = E[T^2] - (E[T])^2$, we can now derive the following.

(1) If $p \leq 1/b^2$, we obtain polynomial scaling for the variance, as $\text{Var}[T] \leq E[T^2] \leq C_n q^2(n)(n+1)$.

(2) For $p > 1/b^2$, the variance scales exponentially in n . To prove this, we establish a lower bound for $\text{Var}[T]$. $\text{Var}[T] \geq C_n q^2(n)(pb^2)^n - C_n^2 q^2(n) [\sum_{i=0}^n (pb)^i]^2 = C_n q^2(n) [(pb^2)^n - C_n [\sum_{i=0}^n (pb)^i]^2] \geq C_n q^2(n) [(pb^2)^n - C_n (n+1)^2 M_n^2] = C_n q^2(n) (pb^2)^n [1 - C_n (n+1)^2 M_n^2 / (pb^2)^n]$, where M_n is the maximum term in the summation $\sum_{i=0}^n (pb)^i$. There are two cases: if $p > 1/b$, $M_n = (pb)^n$, and if $1/b^2 < p \leq 1/b$, $M_n = 1$. In either case, $[1 - C_n (n+1)^2 M_n^2 / (pb^2)^n]$ goes to 1 in the limit $n \rightarrow \infty$, and $\text{Var}[T]$ is bounded below by $(pb^2)^n$ times a polynomial (for sufficiently large n). Since $p > 1/b^2$ by assumption, we have an exponential lower bound.

Next, we establish that the probability distribution is bounded heavy-tailed when $p > 1/b$. That is, the distribution exhibits power-law decay up to run time values of b^n . Set $\epsilon = (1-p)/b$. Then, $C_n p^i \geq \epsilon/b^{i-1}$, since $(1-p) \leq C_n$ for all n and $bp > 1$ by assumption. Now consider $P[T(n) \geq L]$, where L is a value such that $b^{i-1} \leq L < b^i$ for some $i = 1, \dots, n$. It follows that $P[T(n) \geq L] \geq P[T(n) = b^i q(n)] = C_n p^i \geq \epsilon/L$. Thus, we again have power-law decay up to $L < b^n$.

Finally, we observe that we can obtain an expected polytime restart strategy. This can be seen by considering a uniform restart strategy with restart time $q(n)$. We have $P[T(n) = q(n)] = C_n$, so the expected run time is $q(n)/C_n$. In the limit $n \rightarrow \infty$, $C_n = 1-p$; so, the expected run time is polynomial in n .

5 Conclusions

Heavy-tailed phenomena in backtrack style combinatorial search provide a series of useful insights into the overall behavior of search methods. In particular, such phenomena provide an explanation for the effectiveness of random restart strategies in combinatorial search [3, 5, 13]. Rapid restart strategies are now incorporated in a range of state-of-the-art SAT/CSP solvers [12, 11, 1, 9]. So far, the study of such phenomena in combinatorial search has been largely based on the analysis of empirical data. In order to obtain a more rigorous understanding of heavy-tailed phenomena in backtrack search, we have provided a formal analysis of the statistical properties of a series of randomized backtrack search

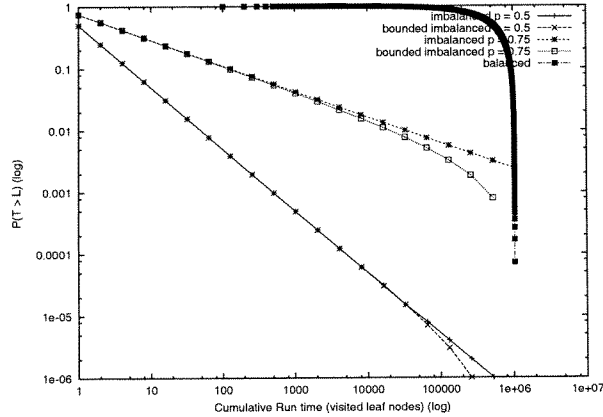


Fig. 4. Example distributions for the balanced, imbalanced and bounded imbalanced models. Parameters: $b = 2$, $n = 20$, $p = 0.5$ and 0.75 .

models: the balanced tree search model, the imbalanced tree model, and the bounded imbalanced tree model. We also studied the effect of restart strategies.

Our analysis for the balanced tree model shows that a randomized backtrack search leads to a uniform distribution of run times (*i.e.*, not heavy-tailed), requiring a search of half of the fringe of the tree on average. Random restarts are not effective in this setting. For the (bounded) imbalanced model, we identified (bounded) heavy-tailed behavior for a certain range of the model parameter, p . The parameter p models “the (in)effectiveness” of the pruning power of the search procedure. More specifically, with probability p , a branching or pruning “mistake” occurs, thereby increasing the size of the subtree that requires traversal by a constant factor, $b > 1$. When $p > 1/b^2$, heavy-tailed behavior occurs. In general, heavy-tailedness arises from a conjugation of two factors: exponentially growing subtrees occurring with an exponentially decreasing probability.

Figure 4 illustrates and contrasts the distributions for the various models. We used a log-log plot of $P(T > L)$, *i.e.*, the tail of the distribution, to highlight the differences between the distributions. The linear behavior over several orders of magnitude for the imbalanced models is characteristic of heavy-tailed behavior [5]. The drop-off at the end of the tail of the distribution for the bounded case illustrates the effect of the boundedness of the search space. However, given the relatively small deviation from the unbounded model (except for the end of the distribution), we see that the boundary effect is relatively minor. The sharp drop-off for the balanced model indicates the absence of heavy-tailedness.

Our bounded imbalanced model provides a good match to heavy-tailed behavior as observed in practice on a range of problems. In particular, depending on the model parameter settings, the model captures the phenomenon of an exponential mean and variance combined with a polynomial expected time restart strategy. The underlying distribution is bounded heavy-tailed.

The imbalanced model can give rise to an effective restart strategy. This suggests some possible directions for future search methods. In particular, it suggests that pruning and heuristic search guidance may be more effective when behaving in a rather asymmetrical manner. The effectiveness of such asymmetric methods would vary widely between different regions of the search space. This would create highly imbalanced search tree, and restarts could be used to eliminate those runs on which the heuristic or pruning methods are relatively ineffective. In other words, instead of trying to shift the overall run time distribution downwards, it may be better to create opportunities for some short runs, even if this significantly increases the risk of additional longer runs.

As noted in the introduction, our imbalanced model is just one particular search tree model leading to heavy-tailed behavior. An interesting direction for future research is to explore other tree search models that exhibit heavy-tailed phenomena.

In our current work, we are also exploring a set of general conditions under which restarts are effective in randomized backtrack search. The long version of the paper, gives a formal statement of such results.

We hope our analysis has shed some light on the intriguing heavy-tailed phenomenon of backtrack search procedures, and may lead to further improvements in the design of search methods.

References

1. R. Bayardo and R. Schrag. Using csp look-back techniques to solve real-world sat instances. In *Proc. of the 14th Natl. Conf. on Artificial Intelligence (AAAI-97)*, pages 203–208, New Providence, RI, 1997. AAAI Press.
2. J. M. Crawford, M. J. Kearns, and R. E. Schapire. The minimal disagreement parity problem as a hard satisfiability problem. Technical report (also in dimacs sat benchmark), CIRL, 1994.
3. C. Gomes, B. Selman, and N. Crato. Heavy-tailed Distributions in Combinatorial Search. In G. Smolka, editor, *Princip. and practice of Constraint Programming (CP97). Lect. Notes in Comp. Sci.*, pages 121–135. Springer-Verlag, 1997.
4. C. Gomes, B. Selman, and H. Kautz. Boosting Combinatorial Search Through Randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 431–438, New Providence, RI, 1998. AAAI Press.
5. C. P. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. of Automated Reasoning*, 24(1–2):67–100, 2000.
6. M. Harchol-Balter, M. Crovella, and C. Murta. On choosing a task assignment policy for a distributed server system. In *Proceedings of Performance Tools '98*, pages 231–242. Springer-Verlag, 1998.
7. I. Jacobs and E. Berlekamp. A lower bound to the distribution of computation for sequential decoding. *IEEE Trans. Inform. Theory*, pages 167–174, 1963.
8. C. M. Li. A constrained-based approach to narrow search trees for satisfiability. *Information processing letters*, 71:75–80, 1999.
9. C. M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 366–371. AAAI Press, 1997.

10. M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of las vegas algorithms. *Information Process. Letters*, pages 173–180, 1993.
11. J. P. Marques-Silva and K. A. Sakallah. Grasp - a search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.
12. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In *Proc. of the 39th Design Automation Conf.*, 2001.
13. T. Walsh. Search in a small world. In *IJCAI-99*, 1999.

Appendix 3 Communication and Computation in DisCSP Algorithms^{*}

Cèsar Fernàndez¹, Ramón Béjar¹, Bhaskar Krishnamachari², and
Carla Gomes²

¹ Departament d'Informàtica i Enginyeria Industrial, Universitat de Lleida
Jaume II, 69, E-25001 Lleida, Spain
{ramon, cesar}@eup.udl.es

² Department of Computer Science, Cornell University
Ithaca, NY 14853, USA
{bhaskar, gomes}@cs.cornell.edu

Abstract. We introduce SensorDCSP, a naturally distributed benchmark based on a real-world application that arises in the context of networked distributed systems. In order to study the performance of DisCSP algorithms in a truly distributed setting, we use a discrete-event network simulator, which allows us to model the impact of different network traffic conditions on the performance of the algorithms. We consider two complete DisCSP algorithms: asynchronous backtracking (ABT) and asynchronous weak commitment search (AWC). In our study of different network traffic distributions, we found that, random delays, in some cases combined with a dynamic decentralized restart strategy, can improve the performance of DisCSP algorithms. More interestingly, we also found that the *active introduction of message delays by agents can improve performance and robustness, while reducing the overall network load*. Finally, our work confirms that AWC performs better than ABT on satisfiable instances. However, on unsatisfiable instances, the performance of AWC is considerably worse than ABT.

1 Introduction

In recent years we have seen an increasing interest in Distributed Constraint Satisfaction Problem (DisCSP) formulations to model combinatorial problems arising in distributed, multi-agent environments [2, 14, 16–18, 20]. There is a rich

^{*} Research partially supported by AFRL, grants F30602-99-1-0005 and F30602-99-1-0006, AFOSR, grant F49620-01-1-0076 (Intelligent Information Systems Institute) and F49620-01-1-0361 (MURI grant on Cooperative Control of Distributed Autonomous Vehicles in Adversarial Environments), CICYT, TIC2001-1577-C03-03 and DARPA, F30602-00-2-0530 (Controlling Computational Cost: Structure, Phase Transitions and Randomization) and F30602-00-2-0558 (Configuring Wireless Transmission and Decentralized Data Processing for Generic Sensor Networks). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government.

set of real-world distributed applications, such as in the area of networked systems, for which the DisCSP paradigm is particularly useful. In such distributed applications, constraints among agents, such as communication bandwidth and privacy issues, preclude the adoption of a centralized approach.

We propose SensorDCSP, a benchmark inspired by one of such distributed applications that arise in networked distributed systems [1, 8]. SensorDCSP is a truly distributed benchmark, a feature not present in many prior benchmark problems used to study the performance of DisCSP algorithms, such as N-Queens and Graph Coloring. SensorDCSP involves a network of distributed sensors simultaneously tracking multiple mobile nodes. The problem underlying SensorDCSP is NP-complete. We show that the SensorDCSP domain undergoes a phase transition in satisfiability, with respect to two control parameters: the level of sensor compatibility and the level of the sensor visibility. Standard DisCSP algorithms on our SensorDCSP domain exhibit the easy-hard-easy profile in complexity, peaking at the phase transition, similarly to the pattern observed in centralized CSP algorithms. More interestingly, the relative strength of standard DisCSP algorithms on SensorDCSP is highly dependent on the satisfiability of the instances. This aspect has been overlooked in the literature due to the fact that, so far, the performance of DisCSP algorithms has been based mainly on satisfiable instances. We study the performance of two well known DisCSP algorithms – asynchronous backtracking (ABT) [18], and asynchronous weak commitment search (AWC) [17]– on SensorDCSP. Both ABT and AWC use agent priority ordering during the search process. While these priorities are static in ABT, AWC allows for dynamic changes in the ordering, and was originally proposed as an improvement over ABT. One of our findings is that although AWC does indeed perform better than ABT on satisfiable instances, its solution time is not as good on unsatisfiable problem instances.

Our SensorDCSP benchmark also allows us to study other interesting aspects specific to DisCSPs that are dependent on the physical characteristics of the distributed environment. For example, while the underlying infrastructure or hardware is not critical in studying CSPs, we argue that this is not the case for DisCSPs in communication networks. This is because the traffic patterns and packet-level behavior of networks, which affect the order in which messages from different agents are delivered to each other, can significantly impact the distributed search process. To investigate these kinds of effects, we implemented our DisCSP algorithms using a *fully distributed discrete-event network simulation environment* with a complete set of communication oriented classes. The network simulator allows us to realistically model the message delivery mechanisms of varied distributed communication environments ranging from wide-area computer networks to wireless sensor networks.

We study the impact of communication delays on the performance of DisCSP algorithms. We consider different link delay distributions. Our results show that the presence of a random element due to the delays can improve the performance of AWC. For the basic ABT, even though link delay deteriorates the performance of the standard algorithm, a decentralized restart strategy that we developed for

ABT improves its solution time dramatically, while also increasing the robustness of solutions with respect to the variance of the network link delay distribution. These results are consistent with results on successful randomization techniques developed to improve the performance of CSP algorithms [4]. Another novel aspect of our work is the introduction of a mechanism for *actively* delaying messages. The active delay of messages decreases the communication load of the system, and, somewhat counter-intuitively, can also decrease the overall solution time.

The organization of the rest of the paper is as follows. In Section 2 we formalize our model of DisCSP. In Section 3 we describe SensorDCSP and model it as a DisCSP. In Section 4 we describe two standard DisCSP algorithms and the modifications we have incorporated into the algorithms. In Section 5 we present our experimental results on the active introduction of randomization by the agents and, in Section 6, we present results on delays caused by different traffic conditions in the communication network. Finally, we present our conclusions in Section 7.

2 Distributed CSPs

In a distributed CSP, variables and constraints are distributed among the different autonomous agents that have to solve the problem. A DisCSP is defined as follows: (1) A finite set of agents A_1, A_2, \dots, A_n ; (2) A set of local (private) CSPs P_1, P_2, \dots, P_n , where the CSP P_i belongs to agent A_i ; A_i is the only agent that can modify the value assigned to the variables of P_i ; (3) A global CSP defined among variables that belong to different agents.

In general in DisCSP algorithms each agent only controls one variable. We extended the single-variable approach by making every agent consist of multiple virtual agents, each corresponding to one local variable. In order to distinguish between communication and computation costs in our discrete event simulator, we use different delay distributions to distinguish between messages exchanged between virtual agents of a single real agent (intra-agent messages) and those between virtual agents of different real agents (inter-agent messages).

3 SensorDCSP - A Benchmark for DisCSP algorithms

The availability of a realistic benchmark of satisfiable and unsatisfiable instances, with tunable complexity, is critical for the study and development of new search algorithms. In the DisCSP literature one cannot find such a benchmark. SensorDCSP, the sensor-mobile problem, is inspired by a real distributed resource allocation problem [13] and offers such desirable characteristics.

In SensorDCSP we have multiple sensors (s_1, \dots, s_m) and multiple mobiles (t_1, \dots, t_n) which are to be tracked by the sensors. The goal is to allocate three distinct sensors to track each mobile node, subject to two sets of constraints: visibility constraints and compatibility constraints. Figure 1 shows an example with six sensors and two mobiles.

Each mobile has a set of sensors that can possibly detect it, as depicted by the bipartite visibility graph in the leftmost panel of Figure 1. Also, it is required that each mobile be assigned three sensors that satisfy a compatibility relation with each other; this compatibility relation is depicted by the graph in the middle panel of Figure 1. Finally, it is required that each sensor only track at most one mobile. A possible solution is shown in the right panel, where the set of three sensors assigned to every mobile is indicated by connecting them to the mobile with the light edges of the figure.

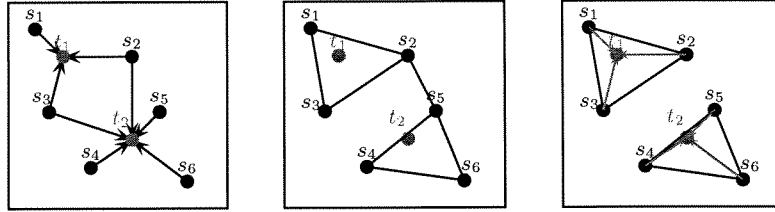


Fig. 1. A SensorDCSP problem instance

This problem is NP-complete since we can reduce it from the problem of partitioning a graph into cliques of size three [1, 6]. However, the boundary case where every pair of sensors is compatible, is polynomially solvable, since we can reduce that case to a feasible flow problem in a bipartite graph [7].

We define a random distribution of instances of SensorDCSP. An instance of the problem is generated from two different random graphs, the visibility graph and the compatibility graph. Apart from the parameters number of mobiles and number of sensors, we also specify a parameter that controls the edge density of the visibility graph (P_v) and a second one that controls the edge density for the compatibility graph (P_c). These parameters specify the independent probability of including a particular edge in the corresponding graph. As these two graphs model the resources available to solve the problem, P_v and P_c control the number of constraints in the generated instances.

We have developed an instance generator for these random distributions that generates DisCSP-encoded instances. We believe that SensorDCSP is a good benchmark problem because of the simplicity of the generator, and because, as we shall show, one can easily generate easy/hard, unsatisfiable/satisfiable instances by tuning the parameters P_v and P_c appropriately.

We encoded SensorDCSP as a DisCSP as follows: each mobile is associated with a different agent. There are three different variables per agent, one for each sensor that we need to allocate to the corresponding mobile. The value domain of each variable is the set of sensors that can detect the corresponding mobile. The intra-agent constraints between the variables of one agent are that the three sensors assigned to the mobile must be different and must be pair-wise compatible. The inter-agent constraints between the variables of different agents are

that a given sensor can be selected by at most one agent. In our implementation of the DisCSP algorithms this encoding is translated to an equivalent formulation where we have three virtual agents for every real agent, each virtual agent handling a single variable.

4 DisCSP algorithms

In the work reported here we considered two specific DisCSP algorithms, Asynchronous Backtracking Algorithm (ABT), and Asynchronous Weak-Commitment Search Algorithm (AWC). We provide a brief overview of these algorithms but refer the reader to [20] for a more comprehensive description. We also describe the modifications that we introduced to these algorithms. As mentioned before, we assume that each agent can only handle one variable. The neighbors of an agent A_i refer to the set of agents that share constraints with A_i .

The **Asynchronous Backtracking Algorithm (ABT)** is a distributed asynchronous version of a classical backtracking algorithm. This algorithm needs a static agent ordering that determines an ordering between the variables of the problem. Agents use two kinds of messages for solving the problem – *ok* messages and *nogood* messages. Agents initiate the search by assigning an initial value to their variables. An agent changes its value when it detects that it is not consistent with the assignments of higher priority neighbors, and so it maintains an agent view, which consists of the variable assignments of its higher priority neighbors.

Each time an agent assigns a value to its variable, it issues the *ok* message to inform its set of lower priority neighbors about this new assignment. When an agent is not able to find an assignment consistent with its higher priority neighbors, it sends a *nogood* message to the lowest priority agent among the agents that have variables in the *nogood*. A *nogood* message consists of a subset of the agent view that does not permit the agent to find a consistent assignment for itself. A *nogood* message causes the receiver agent to record the received *nogood* as a new constraint and to try to find an assignment consistent with its higher priority neighbors and with all the recorded constraints. If the top-priority agent is forced to backtrack, because it cannot fix the problem by asking a higher priority neighbor to change its assignment, this means that the problem has no solution. On the other hand, when the system reaches a state where all agents are happy with their current assignments (no *nogood* messages are generated), this means that the agents have found a solution.

The **Asynchronous Weak-Commitment Search Algorithm (AWC)** can be seen as a modification of the ABT algorithm. The primary differences are as follows. A priority value is determined for each variable, and the priority value is communicated using the *ok* message. When the current assignment is not consistent with the agent view, the agent selects a new consistent assignment that minimizes the number of constraint violations with lower priority neighbors. When an agent cannot find a consistent value and generates a new *nogood*, it sends the *nogood* message to all its neighbors, and increases its priority one unit

over the maximal priority of its neighbors. Then, it finds a value consistent with higher priority neighbors and informs its neighbors with *ok* messages. If no new *nogood* can be generated, the agent waits for the next message.

The most obvious way of introducing randomization in DisCSP algorithms is by randomizing the value selection strategy used by the agents. In the ABT algorithm this is done by performing a uniform random value selection, among the set of values consistent with the agent view and the *nogood* list, every time the agent is forced to select a new value. In the AWC algorithm, we randomize the selection of the value among the values consistent with the agent view and the *nogood* list, and that minimize the number of violated constraints. This form of randomization is analogous to the randomization techniques used in backtrack search algorithms.

A novel way of randomizing the search, relevant in the context of DisCSP algorithms, is by introducing forced delays in the delivery of messages. Delays introduce randomization because the order in which messages arrive to the target agents determines the order in which the search space is traversed. More concretely, every time an agent has to send a message, it follows the following procedure:

1. **With** probability p :
 $d := r$;
else (with probability $(1 - p)$)
 $d := 0$;
2. deliver the message with delay d

By delivering a message with delay d we mean that the agent informs its communication interface that it should wait d seconds before delivering the message through the communication network. The parameter r is the fraction of the mean communication delay added by the agent. In our implementation of the algorithms, this strategy is performed by using the services of the discrete event simulator that allow specific delays to be applied selectively in the delivery message queue of each agent.

We have also developed the following decentralized restarting strategy suitable for the ABT algorithm: the highest priority agent uses a timeout mechanism to decide when a restart should be performed. It performs the restart by changing its value at random from the set of values consistent with the *nogoods* learned so far. Then, it sends *ok* messages to its neighbors, thus producing a restart of the search process, but without forgetting the *nogoods* learned. This restart strategy is different from the restart strategy used in centralized procedures, such as *rand-satz* [4], because the search is not restarted from scratch, but rather benefits from prior mistakes since all agents retain the *nogoods*.

5 Complexity Profiles of DisCSP algorithms on SensorDCSP

As mentioned earlier, when studying distributed algorithms it is important to factor in the physical characteristics of the distributed environment. For exam-

ple, the traffic patterns and packet-level behavior of networks can affect the order in which messages from different agents are delivered to each other, significantly impacting the distributed search process. To investigate these kinds of effects, we have developed an implementation of the algorithms ABT and AWC using the Communication Networks Class Library (CNCL) [5]. This library provides a discrete-event network simulation environment with a complete set of communication oriented classes. The network simulator allows us to realistically model the message delivery mechanisms of varied distributed communication environments ranging from wide-area computer networks to wireless sensor networks.

The results shown in this section have been obtained according to the following scenario. The communication links used for communication between virtual agents of different real agents (inter-agent communication) are modeled as random negative exponential distributed delay links, with a mean delay of 1 time unit. The communication links used by the virtual agents of a real agent (intra-agent communication) are modeled as fixed delay links, with a delay of 10^{-3} time units. We use fixed delay links because we consider that a set of virtual agents work inside a private computation node that allows them to communicate with each other with dedicated communication links. This scenario could correspond to a heavy load network situation where inter-agent delay fluctuations obey to the queuing time process on intermediate systems. The factor of 1000 difference between the two delays reflects that usually intra-agent computation is less expensive than inter-agent communication. In the last section of the paper we will see how different delay distribution models over the inter-agent communication links can impact the performance of the algorithms.

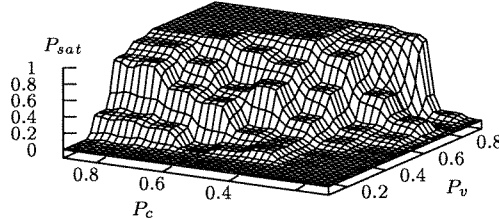


Fig. 2. Ratio of satisfiable instances depending on the density parameter for the visibility graph (P_v) and the density parameter for the compatibility graph (P_c)

For our experimental results, we considered different sets of instances with 3 mobiles and 15 sensors, with every set generated with different values for the parameters P_c and P_v , ranging from 0.1 to 0.9. Every set contains 19 instances, giving a total number of 81 data points. Each instance has been executed 9 times with different random seeds. The results reported in this section were obtained using a sequential value selection function for the different algorithms.

Figure 2 shows the ratio of satisfiable instances as a function of P_c and P_v . When both probabilities are low, the instances generated are mostly unsatisfiable. On the other hand, for high probabilities most of the instances are sat-

isfiable. The transition between the satisfiable and unsatisfiable regions occurs within a relatively narrow range of these control parameters, analogous to the phase transition in CSP problems, *e.g.*, in SAT [10].

Also consistent with general CSP problems, we observe that the phase transition coincides with the region where the hardest instances occur. Figure 3 shows the mean solution time with respect to the parameters P_c and P_v . As can be noted, the hardest instances lie on the diagonal that defines the phase transition zone, with a peak for instances with a low P_c value. The dark and light solid lines overlaid on the mesh depict the location of the iso-lines for $P_{sat} = 0.2$ and $P_{sat} = 0.8$, respectively, as per the phase transition surface of Figure 2. As mentioned before, the SensorDCSP problem is NP-complete only when not all the sensors are compatible between them ($P_c < 1$) [7], so the parameter P_c could separate regions of different mean computational complexity, as in other mixed P/NP-complete problems like 2+p-SAT [10] and 2+p-COL [15]. This is particularly visible in the mean time distribution for AWC in Figure 3.

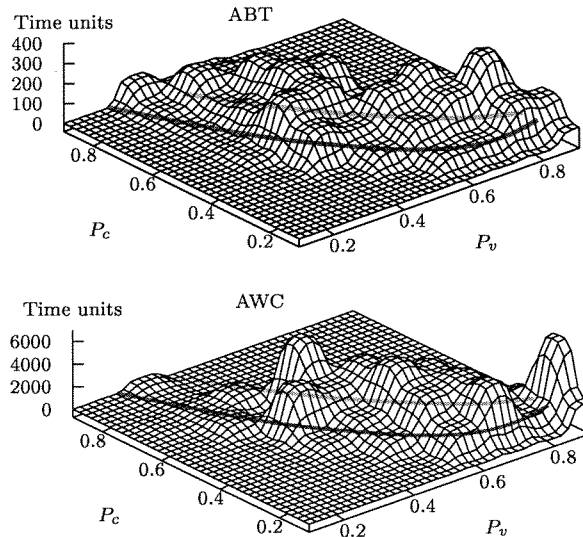


Fig. 3. Mean solution time with respect to P_v and P_c for ABT and AWC algorithms

We observe that the mean times to solve an instance appear to be larger by an order of magnitude for AWC than for ABT. At first glance, this is a surprising result considering that the AWC algorithm is a refinement of ABT and results reported for satisfiable instances in the literature [19, 20] conclude on a better performance for AWC. The explanation for such a discrepancy is the fact that our results deal with both satisfiable and unsatisfiable instances. Our further investigations showed that while AWC does indeed outperform ABT on satisfiable

instances, it is much slower on unsatisfiable instances. This result seems consistent with the fact that the agent hierarchy on ABT is static, while for AWC, such a hierarchy changes during problem solving, taking more time to inspect all the search space when unsatisfiable instances are considered.

5.1 Randomization and restart strategies

In this section we describe experimental results that demonstrate the effect of adding a restart strategy to ABT. The introduction of a randomized value selection function was directly assumed in [19]. In extensive experiments we have performed with our test instances, we noticed that the randomized selection function is indeed better than a fixed selection function. However, as the randomization can introduce more variability in the performance, ABT should be equipped with a restart strategy. We have not defined a restart strategy for AWC, because, as we will see in the last section, the dynamic priority strategy of AWC can be viewed as a kind of built-in partial restart strategy. In the results reported in the rest of the paper both ABT and AWC use randomized value selection functions.

To study the benefits of the proposed restart strategy for ABT, we have solved hard satisfiable instances with ABT with restarts, using different cutoff times. Figure 4 shows the mean time needed to solve a hard satisfiable instance with the corresponding 95% confidence intervals for different cutoff times. We observe clearly that there is an optimal restart cutoff time that gives the best performance. As we will discuss in the last section, when considering the delays of real communication networks, the use of restart strategies becomes a requirement, given the high variance in the solution time due to randomness of link delays in the communication network.

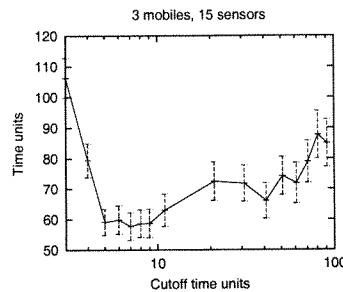


Fig. 4. Mean time to solve a hard satisfiable instance by ABT using restarts with different cutoff times

5.2 Active Delaying of Messages

A novel way of randomizing a DisCSP algorithm corresponds to introducing delays in the delivery of the agents' outgoing messages, as we described in Sec-

tion 4. In this section we describe our experimental results using AWC, where the amount of delay added by the agents is a fraction r (from 0 to 1) of the fixed delay on the inter-agent communication links. In other words, we consider that all the inter-agent communication links have fixed delays, of 1 time unit, in contrast to what we did in the previous sections, because we want to isolate the effect of the delay added by the agents.

Figure 5 shows the results for a hard satisfiable instance from our Sensor-DCSP domain, for different values of p , the probability of adding a delay, and r , the fraction of delay added with respect to the delay of the link. We have that the difference in performance in number of messages can be as high as 3 times between the best case and the worst case. The horizontal plane cutting the surface shows the median time needed by the algorithm when we consider no added random delays ($p = 0, r = 0$). We see that agents can indeed improve the performance by actively introducing some additional random delays, when exchanging messages. We also observe that the performance in number of mes-

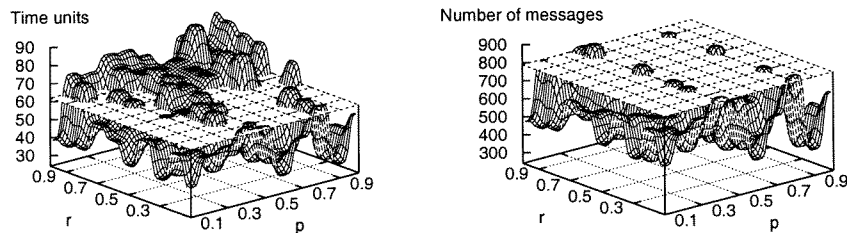


Fig. 5. Median time and number of messages to solve a hard satisfiable instance when agents add random delays in outgoing messages. The horizontal plane represents the median time when no delay is added ($p = 0$)

sages is almost always improved when agents add random delays. Perhaps more surprisingly, in terms of the total solution time, the performance can also improve, if the increase in delay r is not too high. The reason could be the ability of AWC to exploit randomization during the search process due to its inherent restarting strategy.

6 The effect of the communication network data load

As described in the previous section, when working on a communication network with fixed delays, the performance of AWC can be improved, depending on the amount of random delay addition that the agents introduce into the message delivery system. However, in real networks, the conditions of data load present in the communication links used by the agents cannot always be modeled with fixed delay links. It is worthwhile understanding how different communication network environments can impact the performance of the algorithms. In this

section we study the effect produced in the performance of DisCSP algorithms by considering delay distributions corresponding to different traffic conditions.

For the results of Section 5.2 we considered inter-agent communication links with random exponentially distributed delays. To study how exponentially distributed delays affect the performance with respect to fixed delays, we can consider intermediate situations in which some of the inter-agent links have a fixed delay and the rest are exponentially distributed.

Figure 6 shows the median number of messages and time needed by AWC for solving a hard satisfiable instance with 4 mobiles and 15 sensors, when we vary the percentage of inter-agent communication links with a fixed delay. The rest of the inter-agent communication links are assumed to have random exponentially distributed delays.

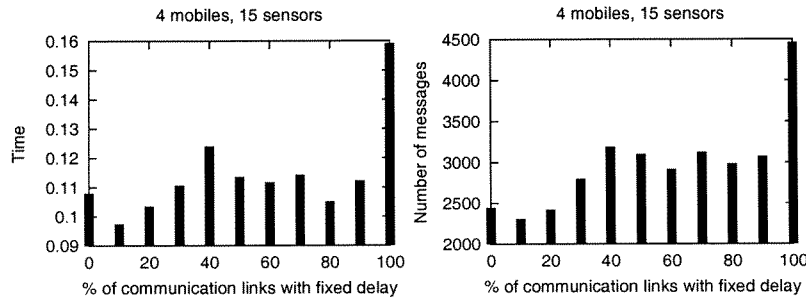


Fig. 6. Median number of messages and time exchanged to solve a hard satisfiable instance by AWC when the data load is not homogeneous among all the inter-agent communication links

The performance of AWC is worst when 100% of the links have a fixed delay, indicating that the conditions of the network clearly affect the performance of the algorithm: a element of randomness in the delay distributions clearly improves the performance of AWC. Observe that we have a clear correlation between the number of messages and time needed, meaning that the increase or decrease in the time needed is mainly because of the change in the number of messages exchanged.

We now examine various link delay distributions that can be used to model communication network traffic. Traditionally, exponential negative distributed inter-arrival times have been used to model data traffic due to their attractive theoretical properties, but in the past decade it has been shown that, although these models are able to capture single user sessions properties, they are no longer suitable for modeling aggregate data links in local or wide area network scenarios[3, 9, 11]. Facing this fact, we simulate network delays according to three different models for the inter-arrival time distribution; the above mentioned exponential negative distribution, the log-normal distribution and the Fractional Gaussian Noise (FGN)[12].

The log-normal distribution is useful to obtain distributions with any desired variance, whereas FGN processes are able to capture crucial characteristics of the Internet traffic as long-range dependence and self-similarity that are not suited by other models. We synthesize FGN from α -stable distributions with parameters $H = 0.75$ and $d = 0.4$,

Figure 7 shows the cumulative density functions (CDF) of time required to solve hard instances for AWC, ABT, and ABT with restarts, when all the inter-agent communication links have delays modeled as fixed, negative exponential, and log-normal, with identical mean and different variances.

Table 1 and 2 show the estimated mean and variance of the number of messages exchanged as well as the solution time for the different cases when the same instance is used for three algorithms.

Delay distribution	Mean			Variance		
	ABT	ABT-rst	AWC	ABT	ABT-rst	AWC
Fixed	$1.8 \cdot 10^5$	$1.2 \cdot 10^5$	$8.2 \cdot 10^2$	$3.6 \cdot 10^{10}$	$1.3 \cdot 10^{10}$	$3 \cdot 10^5$
Negative expon. ($\sigma^2 = 1$)	$1.7 \cdot 10^5$	$1.5 \cdot 10^5$	$3.5 \cdot 10^2$	$2.8 \cdot 10^{10}$	$0.9 \cdot 10^{10}$	$4.5 \cdot 10^5$
Log-normal ($\sigma^2 = 5$)	$2.2 \cdot 10^5$	$1.3 \cdot 10^5$	$3.5 \cdot 10^2$	$5.0 \cdot 10^{10}$	$1.7 \cdot 10^{10}$	$4.8 \cdot 10^5$
Log-normal ($\sigma^2 = 10$)	$2.6 \cdot 10^5$	$1.6 \cdot 10^5$	$3.5 \cdot 10^2$	$7.1 \cdot 10^{10}$	$2.4 \cdot 10^{10}$	$4.9 \cdot 10^5$

Table 1. Statistics estimated from the distributions of number of messages with different inter-agent link delay models

Delay distribution	Mean			Variance		
	ABT	ABT-rst	AWC	ABT	ABT-rst	AWC
Fixed	98	69	53	8562	3600	1230
Negative expon. ($\sigma^2 = 1$)	111	71	28	10945	3947	266
Log-normal ($\sigma^2 = 5$)	157	103	28	21601	8438	288
Log-normal ($\sigma^2 = 10$)	188	131	28	30472	13423	402

Table 2. Statistics estimated from the distributions of time to solve in time units with different inter-agent link delay models

The results in Figure 7 and Tables 1 and 2 show that the delay distributions have an algorithm-specific impact on the performance of the basic ABT and on AWC.

For the basic ABT, on hard instances, the solution time becomes worse when channel delays are modeled by random distributions as opposed to the fixed delay case. The greater the variance of the link delay, the worse ABT performs. However, introducing the restart strategy has the desirable effect of improving the performance of ABT. Furthermore, ABT with restarts is fairly robust and insensitive to the variance in the link delays.

AWC behaves differently from the basic ABT. On hard instances, having randomization in the link delays improves the solution time compared to the fixed delay channel. Further, the mean solution time for AWC is extremely robust to the variance in communication link delays, although the variance of solution time is affected a little bit by this.

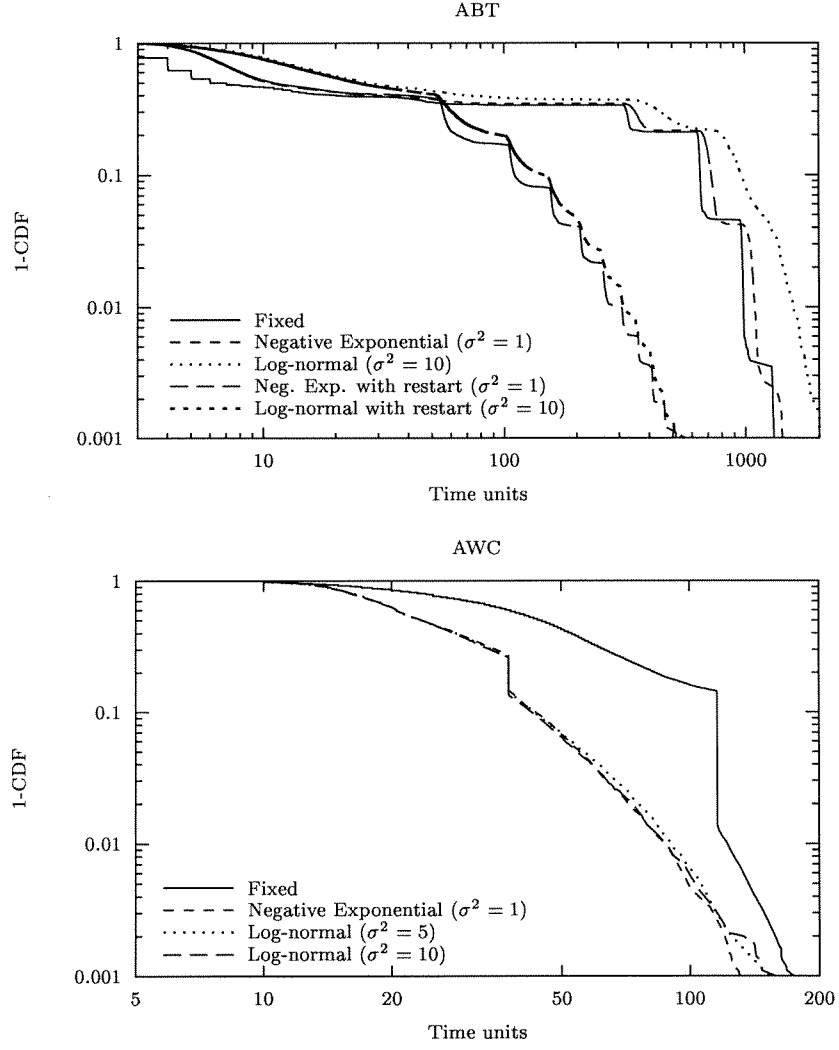


Fig. 7. Cumulative density functions (CDF) of time to solve hard instances for their respective algorithms, AWC, ABT and ABT with restarts under different link delay models

Experiments run with FGN delay models show no significant differences in performance for the three algorithms in relation to other traffic models with the same variance.

In general, we found that on satisfiable instances, AWC always performs significantly better than ABT, even ABT with restart. Thus AWC appears to be a better candidate in situations when most instances are likely to be satisfiable.

7 Conclusions

We introduce SensorDCSP, a benchmark that captures some of the characteristics of real-world distributed applications that arise in the context of distributed networked systems. The two control parameters of our SensorDCSP generator, sensor compatibility (P_c) and sensor visibility (P_v), result in a zero-one phase transition in satisfiability.

We tested two complete DisCSP algorithms, synchronous backtracking (ABT) and asynchronous weak commitment search (AWC). We show that the phase transition region of SensorDCSP induces an easy-hard-easy profile in the solution time, both for ABT and AWC, which is consistent with CSPs. We found that AWC performs much better than ABT on satisfiable instances, but worse on unsatisfiable instances. This differential in performance is most likely due to the fact that on unsatisfiable instances, the dynamic priority ordering of AWC slows the completion of the search process.

In order to study the impact of different network traffic conditions on the performance of the algorithms, we used a discrete-event network simulator. We found that random delays can improve the performance and robustness of AWC. In contrast, on hard satisfiable instances, the performance of the basic ABT deteriorates dramatically when subject to random link delays. However, we developed a decentralized dynamic restart strategy for ABT, which results in an improvement and shows robustness with respect to the variance in link delays. More interestingly, our results also show that the active introduction of message delays by agents can improve performance and robustness, while reducing the overall network load.

These results validate our thesis that when considering networking applications of DisCSP, one cannot afford to neglect the characteristics of the underlying network conditions. The network-level behavior can have an important, algorithm-specific, impact on solution time. Our study makes it clear that DisCSP algorithms are best tested and validated on benchmarks based on real-world problems, using network simulators. We hope our benchmark domain will be of use for the further analysis and development of DisCSP methods.

References

1. R. Béjar, B. Krishnamachari, C. Gomes, and B. Selman. Distributed constraint satisfaction in a wireless sensor tracking system. In *Workshop on Distributed Constraint Reasoning, International Joint Conference on Artificial Intelligence*, Seattle, Washington, August 2001. http://liawww.epfl.ch/~silaghi/proc_wsijcai01.html.

2. S. E. Conry, K. Kuwabara, V. R. Lesser, and R. A. Meyer. Multistage negotiation for distributed constraint satisfaction. *IEEE Transactions on Systems, Man, and Cybernetics (Special Section on DAI)*, 21(6):1462–1477, 1991.
3. M. Crovella and A. Bestavros. Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes. *IEEE Transactions on Networking*, 5(6):835–846, December 1997.
4. C. P. Gomes, B. Selman, and H. A. Kautz. Boosting combinatorial search through randomization. In *AAAI/IAAI*, pages 431–437, 1998.
5. M. Junius, M. Büter, D. Pesch, et al. CNCL. Communication Networks Class Library. Aachen University of Technology. 1996.
6. D. Kirkpatrick and P. Hell. On the complexity of general graph factor problems. *SIAM Journal of Computing*, 12(3):601–608, 1983.
7. B. Krishnamachari. *Phase Transitions, Structure, and Complexity in Wireless Networks*. PhD thesis, Electrical Engineering, Cornell University, Ithaca, NY, May 2002.
8. B. Krishnamachari, R. Béjar, and S. B. Wicker. Distributed problem solving and the boundaries of self-configuration in multi-hop wireless networks. In *Hawaii International Conference on System Sciences (HICSS-35)*, January 2002.
9. W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the Self-Similar Nature of Ethernet Traffic (Extended Version). *IEEE Transactions on Networking*, 2(1):1–15, February 1994.
10. R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic ‘phase transitions’. *Nature*, 400:133–137, July 1999.
11. V. Paxson and S. Floyd. Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995.
12. G. Samorodnitsky and M. S. Taqqu. *Stable Non-Gaussian Random Processes*. Chapman & Hall, 1994.
13. Sanders and Air Force Research Lab. ANTs challenge problem. <http://www.sanders.com/ants/overview-05-09.pdf>, 2000.
14. K. Sycara, S. Roth, N. Sadeh, and M. Fox. Distributed constrained heuristic search. *IEEE Transactions on Systems, Man and Cybernetics*, 21(6):1446–1461, 1991.
15. T. Walsh. The interface between P and NP: COL, XOR, NAE, 1-in-k, and Horn SAT. *APES Report*, APES-37-2002, 2002.
16. M. Yokoo. Weak-commitment search for solving constraint satisfaction problems. In *Proceedings of the 12th Conference on Artificial Intelligence (AAAI-94)*, pages 313–318, 1994.
17. M. Yokoo. Asynchronous weak-commiment search for solving distributed constraint satisfaction problems. In *Proceedings of the First International Conference on Principles and Practice of Constraint Programming (CP-95)*, pages 88–102, 1995.
18. M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *Proceedings of the Twelfth IEEE International Conference on Distributed Computing Systems*, pages 614–621, 1992.
19. M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on Knowledge Data Engineering*, 10(5):673–685, 1998.
20. M. Yokoo and K. Hirayama. Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems*, 3(2):198–212, 2000.